

(NASA-CR-150864) A STUDY OF DIGITAL
HOLOGRAPHIC FILTERS GENERATION. PHASE 2:
DIGITAL DATA COMMUNICATION SYSTEM, VOLUME 1
Report, Mar. 1975 - Dec. 1978 (Mississippi
State Univ., Mississippi State.) 184 p HC

N79-12419

AD9/MF AD1
G3/35 39016

Unclas



ENGINEERING & INDUSTRIAL RESEARCH STATION

ELECTRICAL ENGINEERING/MISSISSIPPI STATE UNIVERSITY

**A STUDY OF DIGITAL HOLOGRAPHIC FILTER GENERATION
PHASE II: DIGITAL DATA COMMUNICATION SYSTEM**

Volume I of II

by
Frank M. Ingels
Chung-Dau Mo



MSSU-EIRS-EE-79-1-VOL-I

COLLEGE OF ENGINEERING ADMINISTRATION

WILLIE L. MCDANIEL, JR., PH.D.
DEAN, COLLEGE OF ENGINEERING

WALTER R. CARNES, PH.D.
ASSOCIATE DEAN

LAWRENCE J. HILL, M.S.
DIRECTOR, ENGINEERING EXTENSION

CHARLES B. CLIETT, M.S.
AEROPHYSICS & AEROSPACE ENGINEERING

WILLIAM R. FOX, PH.D.
AGRICULTURAL & BIOLOGICAL ENGINEERING

JOHN L. WEEKS, JR., PH.D.
CHEMICAL ENGINEERING

ROBERT M. SCHOLTES, PH.D.
CIVIL ENGINEERING

B. J. BALL, PH.D.
ELECTRICAL ENGINEERING

W. H. EUBANKS, M.ED.
ENGINEERING GRAPHICS

FRANK E. COTTON, JR., PH.D.
INDUSTRIAL ENGINEERING

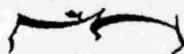
C. T. CARLEY, PH.D.
MECHANICAL ENGINEERING

JOHN I. PAULK, PH.D.
NUCLEAR ENGINEERING

ELDRED W. HOUGH, PH.D.
PETROLEUM ENGINEERING

For additional copies or information
address correspondence to:

ENGINEERING AND INDUSTRIAL RESEARCH STATION
DRAWER DE
MISSISSIPPI STATE UNIVERSITY
MISSISSIPPI STATE, MISSISSIPPI 39762
TELEPHONE (601) 325-2266



Mississippi State University does not discriminate on the grounds of race, color, religion, sex, or national origin.

Under the provisions of Title IX of the Educational Amendments of 1972, Mississippi State University does not discriminate on the basis of sex in its educational programs or activities with respect to admissions or employment. Inquiries concerning the application of these provisions may be referred to Dr. T. K. Martin, Vice President, 610 Allen Hall, Drawer J, Mississippi State, Mississippi 39762, or to the Director of the Office for Civil Rights of the Department of Health, Education and Welfare.



A STUDY OF DIGITAL HOLOGRAPHIC FILTER GENERATION
PHASE II: DIGITAL DATA COMMUNICATION SYSTEM

by

Principal Investigator: Frank M. Ingels

Prepared by: Chung-dau Mo

Submitted by

Mississippi State University
Engineering and Industrial Research Station
Department of Electrical Engineering
Mississippi State, Mississippi
39762

Submitted to

George C. Marshall Space Flight Center
National Aeronautics and Space Administration
Marshall Space Flight Center, Alabama

Under

NASA Contract No. NAS8-31373
For the Period
March 1975 through December 1978

Volume I of II

PRECEDING PAGE BLANK NOT FILMED

(II III)

SUMMARY

This two part volume reports on the second and third phases of the NASA research contract NAS8-31373 initiated March 1975 and culminating in December 1978.

The three phases of this contract are identified by three work statements:

Phase I: Work Statement I dated March 1975

Phase II: Work Statement II dated December 1976

Phase III: Work Statement III dated November 1977.

Work Statement I dealt with the generation of Holographic filters by digital techniques and the work of this phase was reported in final form through the interim final report dated August 1976 (Mississippi State University Report MSSU-EIRS-EE-77-1).

Work Statement II deals with digital data communication over RF channels with both random and bursty error characteristics, i.e. a compound channel. Volume I of the final report dated December 1978 reports on this phase of the contract. In particular a Hybrid decoder was developed and proven by simulation to be superior to a Viterbi Decoder.

Work Statement III deals with a survey of projected coding technology developments in the 1980-1985 time frame. Volume II of the final report dated December 1978 reports on this phase of the contract.

ABSTRACT

The main purpose of this research is to develop an algorithm whose performance approaches that of the optimal maximum likelihood Viterbi decoder in dealing with random errors and yet possesses a burst-error-correcting capability close to that of the optimal B2 decoders. The algorithm developed uses a syndrome detecting logic to direct two decoders, namely, the Viterbi decoder and the algebraic decoder, to assume the decoding load alternatively, depending on the channel characteristics at the moment.

The algebraic decoder used in the hybrid algorithm is an optimal B2 burst corrector suitable for the decoding of nonsystematic convolutional codes. The algebraic decoder is capable of correcting any short burst as long as the total number does not exhaust the storage capacity of the decoder. It can also correct long bursts equal to or less than (constraint length -1) blocks with high probability without having to sacrifice the random-error-correcting capability of the codes. When encountering long bursts which are beyond the error correcting capability of the algebraic decoder, the overall system will outperform the Viterbi decoder because error propagations are suppressed.

The hybrid system works extremely well for compound channels with periodic bursty interferences; however, for random bursty channels in which the required clean guard spaces are not present, the performance of the hybrid system may on occasion become worse than that of the Viterbi decoder.

An empirical study of the performance of the Viterbi decoders in bursty channels was carried out and an improved algebraic decoder for nonsystematic codes was developed. The hybrid algorithm was simulated for the $(2,1)$, $k = 7$ code on a computer using 20 channels having various error statistics, ranging from pure random error to pure bursty channels. The hybrid system outperformed both the algebraic and the Viterbi decoders in every case, except the 1% random error channel where the Viterbi decoder had one bit less decoding error.

TABLE OF CONTENTS

Chapter	Page
SUMMARY	iv
ABSTRACT	v
LIST OF FIGURES	ix
LIST OF TABLES	xi
I. INTRODUCTION	1
INFORMATION TRANSMISSION	1
COMPOUND CHANNELS	6
HISTORICAL BACKGROUND OF CODING THEORY	8
CONCLUSION	12
II. CONVOLUTIONAL CODES	14
THE CODES	14
ALGEBRAIC ENCODING AND DECODING	14
SEQUENTIAL ENCODING AND PROBABILISTIC DECODING	27
SPECIAL CODES - CODES FOR BURST ERRORS	35
CONCLUSION	37
III. PERFORMANCE OF THE VITERBI DECODING ALGORITHM FOR COMPOUND CHANNELS	39
INTRODUCTION	39
METHOD OF STUDY AND SOME PRELIMINARY FINDINGS	40
RESULTS AND FINDINGS OF COMPUTER SIMULATIONS	53
IV. ALGEBRAIC DECODER FOR NONSYSTEMATIC CODES	73
INTRODUCTION	73
PHYSICAL REALIZATION OF CONVOLUTIONAL ENCODERS	75

ALGEBRAIC DECODING OF NONSYSTEMATIC CODES	82
ERROR BOUND AND CHARACTERISTICS OF THE DECODER . .	100
CONCLUSION	103
V. HYBRID DECODER	105
PHILOSOPHY	105
ALGORITHMS	107
COMPUTER SIMULATION	119
EXPERIMENTAL RESULTS AND FINDINGS	123
DISCUSSIONS AND CONCLUSION	129
VI. RECOMMENDATIONS	133
THE GOAL	133
PROBLEMS AND POSSIBLE APPROACHES OF ATTACKING THESE	
PROBLEMS	133
FURTHER IMPROVEMENT OF THE PROPOSED ALGORITHM . .	137
APPENDIX: SIMULATION	138
A: ENCODER	140
B: CHANNEL	141
C: VITERBI DECODER	143
D. ALGEBRAIC DECODER AND SYNDROME DETECTING LOGIC .	145
E. HYBRID DECODER	148
REFERENCES	171

LIST OF FIGURES

Figure	Page
1.1 Block diagram of communication system	7
2.1 Parity check coder	17
2.2 Convolutional encoder	20
2.3 Free distance bounds	26
2.4 A particular (2,1), k=3 convolutional encoder . .	29
2.5 State diagram and trellis diagram	30
2.6 Tree diagram	32
3.1 Convolutional encoder (k=3)	43
3.2 Input and output data sequences	43
3.3 Periodic portion of trellis diagram	44
3.4 Illustrative diagram of decoder operation elements	44
3.5 - 3.8 Illustrative example of the decoding operations	49-52
3.9 Typical print out of the simulation program - Viterbi decoder	54-55
3.10 - 3.14 Performance curves of the decoders simulated	68-72
4.1 Obvious encoder realization	80
4.2 Minimal encoder realization	81
4.3 Encoder for the (2,1), k=7 code	85
4.4 Syndrome tree	92
4.5 Syndrome search logic	93
4.6 Flag logic	96
4.7 Error-correcting decoder	102
5.1 Hybrid decoding flow chart	108

LIST OF FIGURES (Continued)

Figure	Page
5.2 Syndrome detecting logic	109
5.3 Syndrome detecting logic	113
5.4 Syndrome detecting logic	118
5.5 Syndrome detecting logic	120
5.6 Syndrome search logic connection diagram	121
A.1 Probability distribution function of the pseudo-random numbers	149
A.2 Simulation computer program-hybrid system	150-166
A.3 Typical printout-hybrid system	167-170

LIST OF TABLES

TABLE	Page
3.1 Types of decoders investigated	41
3.2 Parameters varied for each type decoder	60
3.3 Parameters tabulated for each type decoder	61
3.4 Overall channel improvement ratio versus type decoder, DCL, NEP, NSW	62-65
3.5 Word error rate improvement versus type decoder - NEP = 10% - 2000 word simulation	66
3.6 Error statistics for 2000 word simulation for 1% and 0.1% error rates	67
4.1 A sequence of decoder operations	89
4.2 Decoder operations in the presence of a long burst .	98
5.1 List of the syndrome patterns of the short bursts .	111
5.2 Comparison of channel performance for different coding schemes	124-128
A.1 Input parameters of the simulation program - hybrid system	149

CHAPTER I

INTRODUCTION

1.1 Information Transmission

Communication is a process by which messages are transferred from one point to another. A message is assumed to be an ordered selection from a known set of symbols. Furthermore, communication is a random process; otherwise there would be no need to communicate. Let X be a random variable whose sample space consists of the m source symbols; $\{x_1, x_2, \dots, x_m\}$. At the destination, one of the destination symbols of the ensemble $\{y_1, y_2, \dots, y_n\}$ will be identified by the signal transmitted. The destination can be represented by a random variable Y . A signal is a physical realization of the message so that transmission through the media, or channel, is possible. This conversion is done by the transmitter. The function of the receiver is to guess the source symbol sent based upon the received symbol y_j and its a posteriori probability $P[x_i|y_j]$ for $i = 1, 2, 3, \dots, m$. The optimal decision rule is that x_k is determined as the source symbol sent if $P[x_k|y_j] \geq P[x_i|y_j]$ for all $i \neq k$.

The assumption that the messages to be sent are chosen from a finite set may not be as restrictive as it seems to be. (1) The difference between very large and infinite becomes very small when dealing with practical systems. For instance, our language uses discrete symbols but it can be used to express nondiscrete feelings and emotions to a fair degree of accuracy. (2) There is no real

communication channel which can convey signals to an arbitrary degree of accuracy; however, the quantization error caused by converting the continuous sample space into discrete sample space can always be made less than the channel errors if the quantization level is fine enough.

In order to make a quantitative analysis of communication systems we may define the information content of a message as the "uncertainty" of that message. The more likely the occurrence of a message, the less information it carries. Therefore the information content of a message is defined as*

$$I_X(x_i) = -\log P(x_i)$$

which is also called the self-information of $X = x_i$. The average self-information over all source symbols is called the entropy of the source, a term borrowed from statistical thermodynamics and used in the same sense.

Similarly the information provided about the event $X = x_i$ by the occurrence of the event $Y = y_j$ may be defined as the mutual information of the joint event $X = x_i$ and $Y = y_j$. This is a measure of how much the occurrence of a particular alternative, say y_j , in the Y ensemble tells us about the possibility of some alternative, say x_i , in the X ensemble. Mutual information is denoted as

$$I_{X;Y}(x_i, y_j) = \log \frac{P_{X|Y}(x_i | y_j)}{P_X(x_i)}$$

*Log x is base 2, not base 10.

The average mutual information over all input and output symbols is denoted as

$$I(X;Y) = \sum_{I=1}^m \sum_{J=1}^n P_{XY}(x_I; y_J) \log \frac{P_{X|Y}(x_I|y_J)}{P_X(x_I)} .$$

As shown earlier in discussing the optimal receiver decision rule, the effect of the transmission is to alter the probability of each possible input from its a priori, i.e. $P(x_I)$, to its a posteriori value, i.e. $P[x_I|y_J]$. The more the channel can improve the a priori probability, the more capable the channel. If the channel is noiseless, after transmission the a posteriori probabilities of all messages would be deterministic. Obviously, maximizing the average mutual information of a channel over all possible input assignments can serve as a measure of the channel capacity, C .

Although $I(X;Y)$ is a function of both the channel and the input assignment, C is a function of only the channel. As a matter of fact, for a symmetric, discrete memoryless channel--i.e., where the occurrence of each output symbol depends only on the corresponding input symbol and does not depend on inputs and/or outputs in the past, and the transition probabilities $P(y_J|x_I)$ of the channel are symmetric--the maximum occurs when the input symbols are equally probable. Therefore just as entropy is a measure of the source demand, channel capacity can be interpreted as the ability of the channel to handle the demand.

If the channel is noiseless, the channel capacity will always be equal to the source entropy, even when the source rate reaches to infinity. If this were true, there would be no problems of communica-

tions. However, nature is not so benevolent; some disturbances are bound to occur. This means the information handling capability is limited for any physical channel; even the source messages are picked from a finite set, i.e. each symbol carries only a finite amount of information. The first resolution is to break the source symbols into smaller units so that each new symbol will carry less information. This is called source encoding. As mentioned earlier the most efficient use of the channel is to make the a priori probabilities of the source symbols equal. This can be taken care of during source encoding by using longer sequences to encode the less probable messages. The message symbols may be broken down to any level necessary with the smallest possible level being one bit; i.e. messages may be encoded using only two symbols, "0" and "1". The level to which message symbols should be broken depends on the channel quality, or the a posteriori probabilities, or the channel capacity. If the channel signal to noise ratio is so strong that it allows the transmission of 4 symbols without a wrong interpretation at the receiving end, then 4 symbols may be used to encode the message symbols. If the 4 symbols are equally probable, then each symbol will carry $I = -\log \frac{1}{4} = \log 4 = 2$ bits of information. Therefore, the channel capacity is fully used and reliable communication is achieved. However, if the channel cannot distinguish 4 levels of signal yet 4 symbols are used to encode the message symbols, reliable communication cannot be achieved. Up to this point it is clear that as long as the channel is not over used, reliable communication is indeed achievable. This is the basis of Shannon's [1948]¹ noisy-channel coding theorem.

In practical communication systems, an attempt is made to send signals at such low power levels that even when the best signaling method, such as antipodal signaling, is used, the receiver may still make mistakes. This means the channel capacity per unit time is below 1 bit per unit time. Therefore, even if the message symbols are broken into the lowest possible unit, i.e. bits, messages still may not be transmitted with confidence. Fortunately, Shannon's theorem does not say that messages cannot be transmitted at a rate lower than 1 bit/unit time. To make the information content of each bit less than 1 bit, redundant bits which do not carry information have to be added or one bit of information has to be distributed to several bits. This is where coding theory comes into play. This second encoding before transmission is called channel encoding to distinguish it from the process called source encoding. Since it is easier to develop codes for binary coding, normally source symbols are encoded into binary digits first and then channel encoded; however, this is not generally necessary. As a matter of fact, some q-ary BCH (Bose-Chaudhuri-Hocquenghem) codes, such as the Reed-Solomon code, have already been developed. Although the vast majority of the error-correcting codes have been designed for additive noise, one-way, memoryless, symmetric channels having orthogonal input signals, work has been done for other types of channels, for instance, channels with memory, channels with nonorthogonal input signals, and asymmetric channels. For practical applications, channels with memory deserve much more attention and even more so in the future.

1.2 Compound Channels

During the discussion of the communication theory, the situation was over simplified by use of a discrete memoryless channel as the model. Although in nature there are channels which can be effectively modeled as additive white Gaussian noise channels (AWGN), for instance the space channels, the vast majority of the real channels do have some degree of memory, i.e. each symbol in the output sequence depends statistically both on the corresponding input and on past inputs and outputs. A physical channel of this nature is a channel whose noises cluster into bursts, i.e. it will have numerous errors in a short time span and be extremely clean at another time. We call the noisy periods the bursts and the clean periods the guard space. A strictly bursty channel, like a random error channel, is a highly idealized model; therefore, the so called compound channel--a channel with both random errors and bursts--becomes a much more realistic model for most physical communication channels. Unfortunately, due to the complexity of its mathematical analysis, few results have been achieved in this area. The existing coding schemes for compound channels lean either to one side or the other or are severely limited to some special situations. This problem is the major concern of the rest of the work.

The reason we regard noise as the only factor which gives a channel memory is that the dependence of message symbols is removed after source encoding. In fact, message symbols are highly dependent in most practical situations, for instance, the probability of an English letter is statistically dependent on the occurrence of the

preceding letters, words on the preceding words, sentences on the preceding sentences However, it is possible to map any message symbol set onto another set of two symbols, for instance "0" and "1", such that the probabilities of these two symbols are statistically independent. Thus the channel can be modeled as the binary symmetric channel (BSC). Reasons for breaking message symbols to this fundamental level are two-fold: (1) q-ary channels perform worse than binary channels in fidelity, although they can transmit data (not information) at a much higher rate; (2) for ease of analysis and applications of error-correcting codes, q-ary codes are complicated to implement.

At this point a block diagram may be constructed to summarize the discussion so far (Figure 1.1).

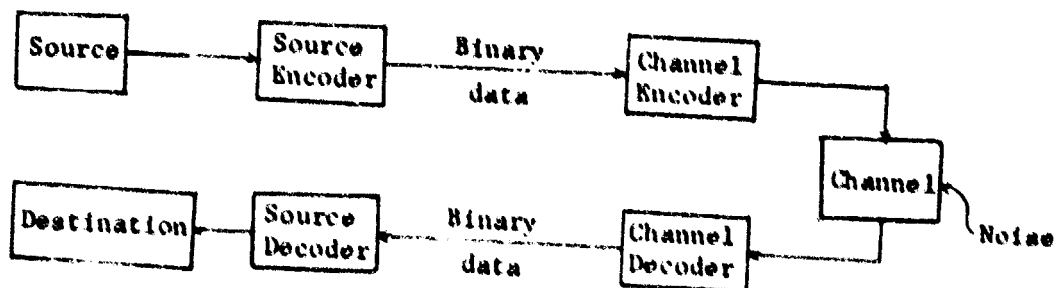


Fig. 1.1 Block diagram of communication system.

For an analog message source a quantizer may be used to convert the analog messages into discrete symbols from a finite set. The ensemble may be called a source alphabet and each sample point a letter. The next step is to source encode these letters into binary digits and then attach redundant bits, i.e. channel encoding, before

sending to the transmitter. The transmitter converts the binary stream into suitable form for transmission over the physical channel; this involves various signaling and digital modulation schemes. At the receiving end the reverse procedure is applied; first, the received waveforms are demodulated, then "channel decoded" to remove errors, "source decoded" to convert the binary stream into source letters and finally the received messages are dumped to data sink, or the user. If analog messages are desired, a digital-to-analog converter may be used. The quantization error can be made as small as desired by decreasing the step size of the quantization level. Normally, the trade-off depends upon cost consideration and the user's needs. This is true for all other designing factors of a communication system, especially the application of error-correcting codes.

1.3 Historical Background of Coding Theory

Strictly speaking, coding theory has its own origin and was not initiated by Shannon. Prior to the publication of Shannon's coding theorem, Hamming had already developed the (7,4) code; as a matter of fact, it was used in the 1948 paper of Shannon¹ as an example. Shortly thereafter, Golay² published his (23,12) code as well as the rather straightforward generalization of Hamming's (7,4) code to all of the other Hamming codes. Even though both were concerned with the problem of reliable communication over noisy channels, the combinatorial, constructive, deterministic viewpoints of Hamming and the probabilistic, statistical, existential viewpoints of Shannon were quite different. However, Shannon's paper certainly added new vitality and insight into the development of coding theory. Only

seven years later, Elias [1955]³ invented convolutional codes, obviously inspired by Shannon's coding theorem. Since then, coding theory has developed in two different directions--the algebraic and the probabilistic schools--and the codes they developed are called algebraic codes (block codes) and convolutional codes.

The existence of algebraic codes has its own right; these codes are highly structured; their decodings are deterministic; they can be analyzed using abstract mathematics. As early as the 50's Muller [1954]⁴, Reed [1954]⁵ and Slepian [1956]⁶ indicated that a large body of knowledge about finite mathematical structures, i.e. finite rings, fields and algebras, could be brought to bear on the coding problem. Soon after, Prange [1957]⁷ proposed the binary cyclic codes which were generated by the primitive polynomials over the Galois field of 2 symbols. The big breakthrough in the development of algebraic codes was the advent of the BCH codes (Bocquenghem [1959]⁸, Bose and Chaudhuri [1960]^{9,10}), the RS (Reed-Solomon [1960]¹¹) codes and the nonbinary BCH (Gorenstein and Zierler [1961]¹²) codes, which are all generated by minimum polynomials of elements from the Galois field $GF(2^m)$. The decoding algorithm of BCH codes was first proposed by Peterson [1960]¹³, and then generalized and refined by Gorenstein and Zierler, Chien, Berlekamp, Forney and Massey. The decoding of BCH codes also requires computations which use Galois field arithmetic. BCH codes, in fact, are a subclass of the cyclic codes which, in turn are a subclass of the general linear block codes. Besides finite algebras, finite geometries were also applied to the coding problem. Geometry codes were first studied by Rudolph [1964]¹⁴ and later extended and generalized by many other coding investigators.

Both "finite projective geometry" and "Euclidean geometry" were applied in the construction of these codes. The development of geometry codes was motivated by a special decoding method for a subclass of the block codes, including some of the BCH codes, called majority logic decoding. Majority logic decodable codes are orthogonalizable codes. Some of the examples are the (15,7) BCH code, maximum-length codes, difference-set codes, the $(2^m, 2^m - m - 1)$ Hamming code, the $(2^m - 1, m + 1)$ BCH codes, the (31,16) BCH code and the two types of geometry codes mentioned above, with the Reed-Muller codes, as the best known Euclidean geometry codes. Another important decoding method for cyclic codes is the modified Meggitt decoder, called error-trapping decoding, which can be applied to almost any cyclic codes. Although this decoding method enjoys a very simple combinational logic circuit, it loses error-correcting capability for long, high-rate codes.

The algebraic codes, though they enjoy beautiful structural properties, do not approach an arbitrary small error probability for asymptotically long block lengths for rates less than channel capacity as promised by Shannon's coding theorem. While the algebraic codes are asymptotically weak, it was known that the error probability of sequential decoders, invented by Wozencraft and Reiffen [1961]¹⁵ and subsequently refined by Fano [1963]¹⁶ for the decoding of convolutional codes, could be made to approach zero rapidly as long as the transmission rate was below the computational cutoff rate, R_{comp} which, in general, is a substantial fraction of the channel capacity. However, this barrier was further removed by the advent of the Viterbi algorithm [1967].¹⁷ Both the sequential

algorithm and the Viterbi algorithm map the received sequence to the most likely source sequence, i.e. maximum likelihood decoding, while the latter is not bounded by R_{comp} and thus is optimal in the sense of Shannon's coding theorem. Since the number of comparisons required per decoded bit for a Viterbi decoder increases exponentially with constraint length k , it is impractical to attempt to achieve a small probability of error merely by increasing the constraint length. Therefore, it is imperative to find "good" convolution codes. Costello [1974]¹⁸ defined "good" codes as codes that have optimal free distance, d_{free} , a term he invented to represent the entire distance between codewords from the encoder. As a result, he found that the chance of an optimal d_{free} code being systematic is very slim and nearly all the nonsystematic codes are better codes. Due to the "non-algebraic" nature of their structures, convolutional codes are extremely difficult to analyze; consequently, very little is known about their structural properties. This is probably the main drawback of this class of codes. Traditionally, the construction of convolutional codes was done by trial and error and was limited to systematic codes. For nonsystematic codes, so far the only known way to find good codes is to make an exhaustive search of all possible codes, using an educated routine proposed by Bahl et. al.¹⁹

Convolutional codes can be truncated into blocks of fixed length and then decoded algebraically. Although the random-error-correcting capability would be reduced, algebraic decoders are very attractive for burst-error-corrections. Unfortunately, with the exception of the decoder proposed by Robert W. Boyd [1976]²⁰, all the known

algebraic decoding algorithms are for systematic codes. Boyd's decoder used the idea of multiple parity-check and was shown capable of correcting short bursts of one block. Therefore, it is highly desirable to develop further algebraic decoding algorithms for non-systematic convolutional codes to handle much longer bursts. This is one of the major efforts of this dissertation. Clearly if such an algebraic algorithm could be achieved, then with proper interfacing of it with the Viterbi algorithm, a hybrid system could be developed with a performance equal to that of the Viterbi decoder for the random errors and as powerful as the algebraic decoder itself for bursts. For intermediate situations, that is, in the compound channels, the hybrid system should certainly outperform either of the decoders alone.

1.4 Conclusion

A heuristic interpretation of Shannon's coding theorem was attempted and a historical background of the coding development was briefly introduced. This chapter is not intended to be tutorial but only serves to orient the readers to the discussions in the following chapters and make the discussions more comprehensive.

Coding for compound channels does have its "information theory" foundation. Shannon's noisy-channel coding theorem is a very general fundamental theorem. It applies both to memoryless channels and to channels with memory, and can be generalized to nondiscrete channels as well. There are also codes developed for strictly bursty channels and for compound channels, however, most of them are block codes and the known convolutional codes for channels with memory are strictly

for bursty channels; therefore, their random error correcting capabilities are greatly sacrificed. Other techniques such as interlacing and concatenation also have their shortcomings. Interlacing achieves burst error correction at the price of random error correction while concatenation involves using two codes, a process which, of course, introduces rate loss. In any case, none of the existing coding schemes has a performance approaching that of the optimal Viterbi decoders for random errors and yet behaves as an optimal B2 bursts corrector when dealing with clustered errors. It is the purpose of this work to attack this goal. As will be shown in the following discussions, though there are problems not completely solved, some very promising results have been produced and a coding scheme which outperforms other known schemes in a large selection of practical communication channels has been accomplished. It is felt that, at least in principle, an optimal decoding algorithm for compound channels as defined above is achievable. Should such a scheme be realized, coding applications would not be limited to a few special type of channels. Its value and impact to practical communications could be very significant.

CHAPTER II

CONVOLUTIONAL CODES

2.1 The Codes

Convolutional (or recurrent) codes can be regarded either as a subclass of the linear block (or parity check) codes or as linear recurring sequences. Taking the first viewpoint, convolutional codes differ from block codes only in that the present code block depends not only on the corresponding input block but also on the previous $k - 1$ blocks, where k is termed the encoder's constraint length. Taking the second viewpoint, however, convolutional codes are nothing more than the recurring sequences produced by finite state sequential machines and can be analyzed as such.

2.2 Algebraic Encoding and Decoding

Encoding of block codes can most easily be comprehended as a mapping process from a k -dimensional vector space to the subspace of an n -dimensional space, where $n > k$. The optimal mapping rule is to let the minimum distance between any two members in the range space, i.e. code space, be the maximum achievable. Members in the range space are called code words, and their distance is called Hamming distance. For binary codes, the n -space is over $GF(2)$ and the distance between $\underline{u}$ and $\underline{v}$, $d(\underline{u}, \underline{v})$, is defined as the number of components in which they differ.

A shorthand notation for linear vector space is the matrix notation. We may let the k basis vectors (each is a n -tuple) be the k rows of a $k \times n$ matrix G , called generator matrix.

$$\underline{G} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & & \vdots \\ g_{k,1} & g_{k,2} & \cdots & g_{k,n} \end{bmatrix}$$

Then encoding becomes the postmultiplication of the message blocks (k -tuples) by the generator matrix so that all code vectors are linear combinations of the basis vectors that spans the code space.

It is known that for any $k \times n$ matrix $\underline{G}$ with rank k , there exists a $(n-k) \times n$ matrix $\underline{H}$ with rank $(n-k)$ such that the row space of $\underline{G}$ is the null space of $\underline{H}$ or vice versa, $\underline{H}$ is called the parity check matrix.

$$\underline{H} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{n-k} \end{bmatrix} = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,n} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,n} \\ \vdots & \vdots & & \vdots \\ h_{n-k,1} & h_{n-k,2} & \cdots & h_{n-k,n} \end{bmatrix}$$

Decoding is simply the postmultiplication of the received code word $\underline{Y}$ by $\underline{H}$. Since every member in $\underline{G}$, i.e. $\underline{g}$, is orthogonal to every member in $\underline{H}$, if the received $\underline{r}$ is indeed a member of $\underline{G}$, the product would be zero, i.e. $\underline{r} \underline{H} = \underline{g} \underline{H} = \underline{0}$, otherwise, the product would be nonzero, i.e. $\underline{r} \underline{H} = (\underline{g} + \underline{e}) \underline{H} = \underline{g} \underline{H} + \underline{e} \underline{H} = \underline{e} \underline{H} \neq \underline{0}$.

This nonzero vector is called the syndrome vector because it provides some information about the error vector $\underline{e}$.

In order to find $\underline{H}$, $k \times (n-k)$ simultaneous equations, i.e. $\underline{g}_I \underline{h}_J = 0$ for $I = 1, \dots, k$, $J = 1, \dots, n-k$, have to be solved. This involves a tremendous amount of work for large k and $n-k$, and unfortunately this is exactly how a "good code" should be. However, the simplicity of GF(2) arithmetic provides an easy way out.

That is, let $\underline{G}$ contain an identity matrix:

$$\underline{G} = \left[\begin{array}{c|cccc} & P_{11} & P_{12} & \dots & P_{1,n-k} \\ I_k & P_{21} & P_{22} & \dots & P_{2,n-k} \\ & \vdots & \vdots & & \vdots \\ & P_{k1} & P_{k2} & \dots & P_{k,n-k} \end{array} \right] ; \text{ or } \underline{G} = [\underline{I}_k : \underline{P}]$$

then we can form:

$$\underline{H} = \left[\begin{array}{cccc|c} P_{11} & P_{21} & \dots & P_{k1} & \\ P_{12} & P_{22} & \dots & P_{k2} & \\ \vdots & \vdots & & \vdots & \\ P_{1,n-k} & P_{2,n-k} & \dots & P_{k,n-k} & \end{array} \right] I_{n-k} ; \text{ or } \underline{H} = [P^T : I_{n-k}]$$

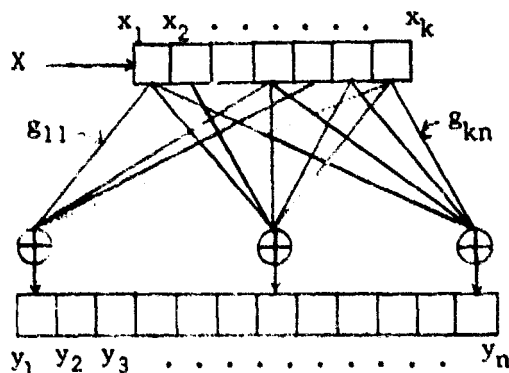
By so doing, it can be easily seen that

$$\underline{G} \cdot \underline{H}^T = [\underline{I} : \underline{P}] \cdot \begin{bmatrix} \underline{P} \\ \underline{I} \end{bmatrix} = [\underline{I} \underline{P} + \underline{P} \underline{I}] = [\underline{P} + \underline{P}] = [0]$$

Code words produced by this type of generator matrix always contain their message words explicitly and therefore are called systematic

codes. Since the additional constraint of containing an identity sub-matrix in the generator matrix imposes a severe limitation on the freedom of choosing good codes, systematic codes normally do not give optimal Hamming distances. Unfortunately, most existing algebraic decoding schemes are for this type of codes.

Physical realization of parity check codes can be achieved with a k -stage shift register connected to an n -stage shift register in parallel through some "EXCLUSIVE OR" gates. The encoder input sequence $\underline{X}$ is shifted into the k -stage shift register k bits at a time and outputs through those "EXCLUSIVE OR" s, or Mod-2 adders, and fed into the n -stage shift registers. The content of the n -stage shift registers is the encoded block of the corresponding source block. The connection vectors between the two shift registers are equal exactly to the k rows of the generator matrix, or the k n -tuples that form the basis of the code space.



If $g_{ij} = 1$, there is a connection between x_i and y_j , otherwise there is no connection.

Fig. 2.1 Parity check coder.

For a low rate ($1/n$) convolutional code the only difference is that instead of shifting the whole block of k bits in, only one bit

is shifted in at each time unit. As a result, we may regard the whole source sequence $\underline{X}$ as a semi-infinite source block and the generator matrix as a semi-infinite matrix:

$$\underline{G} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ \downarrow \\ \infty \end{bmatrix} = \begin{bmatrix} g_1 & \dots & g_k & 0 & 0 & \rightarrow \infty \\ 0 & g_1 & \dots & g_k & 0 & \rightarrow \infty \\ & & & \downarrow & & \\ & & & \infty & & \end{bmatrix}$$

where the g_i 's for $i=1,2,\dots,k$ are equivalent exactly to the g_i 's of the corresponding block code.

This formulation can be easily seen from Fig. 2.1. As x_1 moves along the upper shift register, it will eventually occupy every stage of the register and the same is true for all other source bits except that their occupation of the same stage is delayed one time unit successively. This is why we may visualize x_1 , as being multiplied by $g_1, g_2, g_3, \dots$ and the same is true for $x_2, x_3, \dots, x_L$ except that the multiplications are delayed one time unit, or n bits, for each successive source bit. For convolutional codes, normally a commutator is used to sample the outputs of the $n \bmod 2$ adders and then to send the encoded sequence out for transmission.

This analogous formulation of the convolutional codes not only reveals some of the algebraic properties it inherits from the linear block codes such as the property of linearity, but also provides a decoding method similar to that of the block codes. That is, if the code is systematic, a parity check matrix $\underline{H}_\infty$ can easily be found; therefore, the same decoding procedure as that of the block codes can be applied.

The block code model of convolutional codes also reveals their power and limitations. Due to their recurrent nature, convolutional

codes allow the entire source sequence, say L bits, to be encoded as one whole block with very simple encoding circuitry or small storage requirement. Superficially, it seems that the ultimate random coding bound $P[\epsilon] < 2^{-N[R'_0 - R_n]}$ could be achieved with $P[\epsilon] \rightarrow 0$, when $N = (L + k)n \rightarrow \infty$. However, the proof that this bound is valid depends strongly on freedom to choose the block-coder connection vectors arbitrarily. The additional constraint of being recurrent exactly destroys such freedom and limits the choice of good codes. Indeed, if L is increased while k , the encoder constraint length, is held fixed, error probability cannot be expected to approach zero as $N \rightarrow \infty$. On the other hand, it is reasonable to anticipate a bound on error probability that decreases exponentially with a linear increase in encoder's constraint length k . As will be shown later, merely increasing k will increase the decoding complexity exponentially for the optimal algorithm. Even so there are limitations; the convolutional codes are still, by far, the best known codes in practical situations because of their semi-infinite block length nature.

So far, we have only considered the low rate $1/n$ codes. For the general ℓ/n codes, all we have to do is to use ℓ sets of the construction as shown in Fig. 2.1, and the corresponding outputs of the ℓ k -stage shift registers are mod-2 added together before being sampled, Fig. 2.2.

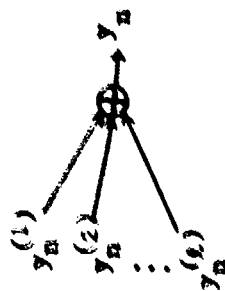
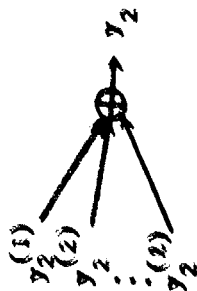
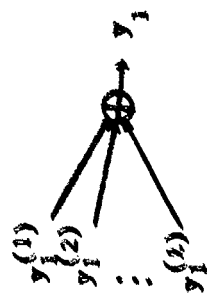
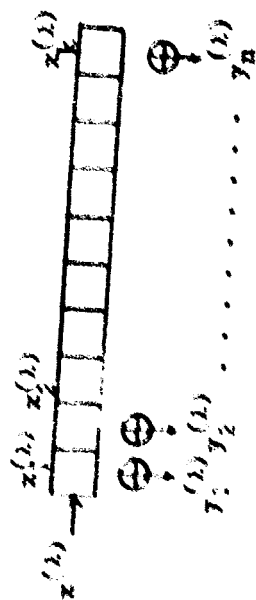
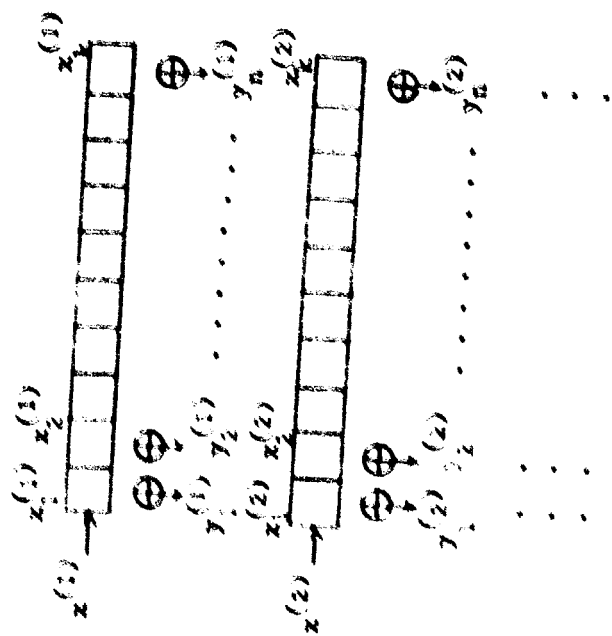


Fig. 2.2 Convolutional encoder.

Therefore, all that have to be specified are the g_{ij} 's for $j > k$.

A more comprehensive set of notations can be borrowed from Shu Lin's book.²¹ Let

$$\underline{G}_0 = \begin{bmatrix} g_0(1,1) & g_0(1,2) & \dots & g_0(1,n) \\ g_0(2,1) & g_0(2,2) & \dots & g_0(2,n) \\ \vdots & \vdots & & \vdots \\ g_0(t,1) & g_0(t,2) & \dots & g_0(t,n) \end{bmatrix} = \begin{bmatrix} g_{11}^{(1)} & g_{12}^{(1)} & \dots & g_{1n}^{(1)} \\ g_{11}^{(2)} & g_{12}^{(2)} & \dots & g_{1n}^{(2)} \\ \vdots & \vdots & & \vdots \\ g_{11}^{(t)} & g_{12}^{(t)} & \dots & g_{1n}^{(t)} \end{bmatrix}$$

$$= [\underline{I} : \underline{P}_0]$$

$$\underline{G}_1 = \begin{bmatrix} g_1(1,1) & g_1(1,2) & \dots & g_1(1,n) \\ g_1(2,1) & g_1(2,2) & \dots & g_1(2,n) \\ \vdots & \vdots & & \vdots \\ g_1(t,1) & g_1(t,2) & \dots & g_1(t,n) \end{bmatrix} = \begin{bmatrix} g_{21}^{(1)} & g_{22}^{(1)} & \dots & g_{2n}^{(1)} \\ g_{21}^{(2)} & g_{22}^{(2)} & \dots & g_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ g_{21}^{(t)} & g_{22}^{(t)} & \dots & g_{2n}^{(t)} \end{bmatrix}$$

$$= [\underline{0} : \underline{P}_1]$$

$$\underline{G}_{k-1} - \underline{G}_T = \begin{bmatrix} g_T(1,1) & g_T(1,2) & \dots & g_T(1,n) \\ g_T(2,1) & g_T(2,2) & \dots & g_T(2,n) \\ \vdots & \vdots & & \vdots \\ g_T(t,1) & g_T(t,2) & \dots & g_T(t,n) \end{bmatrix} = \begin{bmatrix} g_{k1}^{(1)} & g_{k2}^{(1)} & \dots & g_{kn}^{(1)} \\ g_{k1}^{(2)} & g_{k2}^{(2)} & \dots & g_{kn}^{(2)} \\ \vdots & \vdots & & \vdots \\ g_{k1}^{(t)} & g_{k2}^{(t)} & \dots & g_{kn}^{(t)} \end{bmatrix}$$

$$= [\underline{0} : \underline{P}_{T-1}]$$

- where
- (1) $\underline{I}$ is a $l \times l$ identity matrix
 - (2) $\underline{0}$ is a $l \times l$ zero matrix
 - (3) $\underline{P}_r$ is a $l \times (n-l)$ matrix of the following form

$$\underline{P}_r = \begin{bmatrix} g'_r(1,1), & g'_r(1,2), & \dots & g'_r(1,n-l) \\ g'_r(2,1), & g'_r(2,2) & \dots & g'_r(2,n-l) \\ \vdots & \vdots & & \vdots \\ g'_r(l,1), & g'_r(l,2), & \dots & g'_r(l,n-l) \end{bmatrix}$$

With such notations the generator matrix of systematic (n,l) convolutional codes is of the following form:

$$\underline{G}_\infty = \begin{bmatrix} \underline{I} & \underline{P}_0 & \underline{0} & \underline{P}_1 & \underline{0} & \underline{P}_2 & \dots & \underline{0} & \underline{P}_{k-1} & \text{zeros} \rightarrow \infty \\ \underline{0} & \underline{I} & \underline{P}_0 & \underline{0} & \underline{P}_1 & \underline{0} & \underline{P}_2 & \dots & \underline{P}_{k-2} & \underline{0} & \underline{P}_{k-1} & \text{zeros} \rightarrow \infty \\ \underline{0} & \underline{0} & \underline{I} & \underline{P}_0 & \dots & \dots & \dots & \underline{0} & \underline{P}_{k-2} & \underline{0} & \underline{P}_{k-1} & \text{zeros} \rightarrow \infty \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \text{zeros} & & & & & & & & \underline{I} & \underline{P}_0 & \underline{0} & \underline{P}_1 & \underline{0} & \underline{P}_2 \dots \\ & & & & & & & & & \downarrow & & & & \\ & & & & & & & & & \infty & & & & \end{bmatrix}$$

and the parity check matrix $\underline{H}_\infty$ is:

$$\begin{array}{c}
 \begin{array}{cccccc}
 \underline{P}_0^T & \underline{I} & & & & \longrightarrow \text{zeros} \longrightarrow \infty \\
 \underline{P}_1^T & \underline{0} & \underline{P}_0^T & \underline{I} & & \longrightarrow \text{zeros} \longrightarrow \infty \\
 \underline{P}_2^T & \underline{0} & \underline{P}_1^T & \underline{0} & \underline{P}_0^T & \underline{I} \longrightarrow \text{zeros} \longrightarrow \infty \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 \underline{P}_{k-1}^T & \underline{0} & \underline{P}_{k-2}^T & \underline{0} & & \\
 + & & \underline{P}_{k-1}^T & \underline{0} & \underline{P}_{k-2}^T & \underline{0} \\
 \text{zeros} & + & & \underline{P}_{k-1}^T & \underline{0} & \\
 + & & \text{zeros} & + & & \\
 + & + & & \text{zeros} & & \\
 + & + & + & & & \\
 \infty & \infty & \infty & & & \infty
 \end{array}
 \end{array}
 =
 \begin{array}{c}
 \begin{array}{cccccc}
 \underline{P}_1^T & \underline{0} & \underline{P}_0^T & \underline{I} & \longrightarrow \text{zeros} \longrightarrow \infty \\
 \underline{P}_2^T & \underline{0} & \underline{P}_1^T & \underline{0} & \underline{P}_0^T & \underline{I} \longrightarrow 0 \infty \\
 + & & & & & \\
 + & & & & & \\
 \text{zeros} & & & & & \\
 + & & & & & \\
 & & & & & \infty
 \end{array}
 \end{array}$$

where $\underline{I}$ is an $(n - \ell) \times (n - \ell)$ identity matrix and $\underline{0}$, an $(n - \ell) \times (n - \ell)$ zero matrix. With both $\underline{G}_\infty$ and $\underline{H}_\infty$ defined, encoding and decoding can be performed much as they are in the block codes; the size of these matrices being semi-infinite are not as formidable as they seem. Due to the recurrent nature of these matrices, the only information needed is the $\underline{P}_r$'s. They may be grouped together as the so called sub-generator:

$$g(i, j) = (g_0(i, j), g_1(i, j) \dots g_{k-1}(i, j)) \quad \text{where } k-1 = r$$

$$\text{for } i = 1, 2, \dots, \ell, \quad j = 1, 2, \dots, n - \ell$$

Algebraic decoding of convolutional codes has the following advantages:

(1) It involves only multiplications of the received sequence and the parity check matrix, therefore, it enjoys the operational speed advantage and simple circuitry.

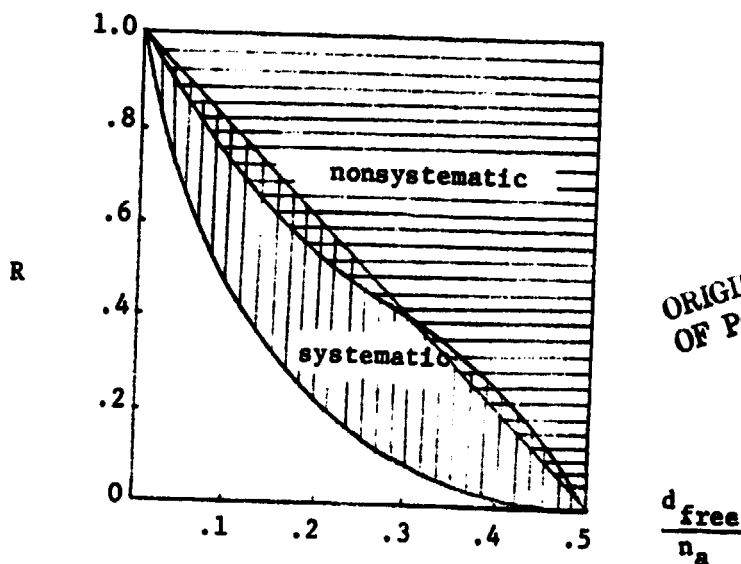
(2) It does not propagate error effects to more than $(k-1)$ time units.

(3) Its burst correcting capability can be made better than other decoding algorithms.

However, since the decoding is performed n digits at a time, it cannot fully use the d_{free} as the probabilistic algorithm, consequently, its random error correcting ability is greatly impaired.

Just as with block codes, the error correcting capability for convolutional codes also depends on the minimum distance between codewords. Due to the semi-infinite nature of these codes, the entire distance between codewords available from the encoder is defined as free distance, d_{free} , to distinguish it from the term minimum distances, d_{min} , which is used in the same sense except that it is a measure of the distance when algebraic decoding is applied. The d_{min} is the useful d_{free} when the received sequence is truncated, therefore, $d_{\text{min}} < d_{\text{free}}$. This is exactly why algebraic decoders are inferior to probabilistic decoders in random error corrections. Another important consideration is that, just as with block codes, systematic encoding does not, in general, give optimal d_{free} convolutional codes. This fact was first discovered by Costello [1974].¹⁸ He has been able to obtain bounds on d_{free} per

transmitted bit $\frac{d_{\text{free}}}{n_a}$ for several types of convolutional codes. Fig. 2.3 illustrates four of the bounds which he obtained. It is clear that in almost any case a nonsystematic convolutional code will be a more powerful error corrector than a systematic code.



ORIGINAL PAGE IS
OF POOR QUALITY

Fig 2.3 Free distance bounds.¹⁸

As will be discussed later, the major part of this work is to develop an algorithm which allows a convolutional code to be decoded by two decoders, an algebraic decoder and a probabilistic decoder, interfaced together. In order to achieve maximum performance, an algebraic decoder that allows the decoding of nonsystematic codes is deemed necessary, for otherwise its random error correcting capability will be severely limited. Unfortunately, no work has been done in the development of algebraic decoding for nonsystematic codes, except that Robert W. Boyd²⁰ proposed an algorithm which is capable

of correcting short bursts of one block long, i.e., n bits. In Chapter IV, this algorithm will be modified and extended to exploit fully the error correcting capability of the codes used. A good code should be able to behave like an optimal B2 burst corrector, and it will be shown that this is indeed so for the example $(2,1)$, $k = 7$ code.

2.3 Sequential Encoding and Probabilistic Decoding

Convolutional encoders can also be modeled as finite state machines and realized as such. This fact can easily be seen from Fig. 2.4. As mentioned previously, for a convolutional code, the code block depends not only upon the present input bit but also the previous $k-1$ bits. Since there are 2^{k-1} possible patterns for the previous $k-1$ bit streams, we may regard a constraint k encoder as having 2^{k-1} states. The output of the encoder depends on both the present input bit and the present state of the machine, i.e. the encoder. In fact, convolutional codes are mainly applications of linear sequential circuits. Other applications of sequential circuits are much simpler. For instance, we may want to realize a sequential machine that gives us a prescribed output sequence for a certain input history; output sequences of other input paths are ignored as long as they do not interfere in the generation of the prescribed sequence. However, for the encoding of convolutional codes, we must consider all the possible input sequences, and the function of the machine is to map the input sequences into the output sequences in such a way that the minimum distance between any two output sequences be the maximum possible, a formidable task.

Therefore, the present methods of finding "good codes" or "good machines" is limited to being empirical and analytical. No synthesis procedure has been developed yet. This is the major difference between the application of linear sequential circuits to convolutional codes and to other purposes. Until more is known about the characteristics of the finite state machines and the structure properties of their output sequences, there probably will not be an important breakthrough.

The decoding of convolutional codes amounts to nothing more than the recovery of the input sequence from the knowledge of the received sequence. The received sequence is the output sequence contaminated by noises during transmission or by erasures during storage. Like block codes, the optimal decoding rule is to map the received sequence to the most likely source sequence. However, because the received sequence is semi-infinitely long, it is not practical to wait until the entire sequence is received before decoding. If this were done, the amount of storage and number of computations required would be beyond our imagination. Fortunately, convolutional codes are not randomly chosen block codes with arbitrary block length; they are recurrent codes. The complexity of convolutional codes depend on their constraint length k . For an $\frac{1}{n}$ code, the encoder has 2^{k-1} states, and the total possible output patterns are 2^k for binary inputs. Thus, intuition tells us that the additional error probability caused by considering truncated received sequences for MLD (maximum likelihood decoding) should be decreasing exponentially with increasing truncation length--this is exactly so. Forney²² used random coding arguments to show that on the average, a

truncation length of about $5(k-1)$ bits will result in an additional error probability that is comparable to the MLD error probability. Hemmati and Costello²³ derived an upper bound and a formula to calculate the bit error probability due to truncation of the received sequence for MLD. They found that, at least for low values of P , if the received sequence is truncated to length T such that $d_T > d_{\text{free}}$, the truncation error probability will be insignificant compared to the MLD error probability, where d_T is the minimum distance between the correct sequence and all other possible sequences at length T . Normally, T is only a few times the encoder's constraint length k .

The sequential or finite state machine formulation of the convolutional codes can best be illustrated with an example. Suppose a rate $\frac{1}{n}$ code is to be implemented with the circuit shown in Fig. 2.4.

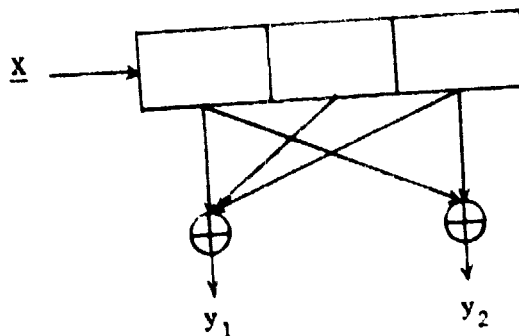


Fig. 2.4 A particular $(2,1)$, $k = 3$ convolutional encoder.

convolutional encoders can always be realized with shift registers of finite stages and some "EXCLUSIVE OR's".

As mentioned earlier, the optimal decoding rule is to map the received sequence to the most likely source sequence, i.e. maximum likelihood decoding. This is the basic principle underlying all decoding methods; it applies to block codes as well as convolutional codes. For linear block codes, MLD simplifies to the mapping of the received vector to the code vector which differs from it in the least number of places among all other vectors in the code space. Though convolutional codes can be treated the same way as linear block codes by truncating the received sequence into short blocks of a fixed number of bits, i.e. algebraic decoding, their semi-infinite block length advantage would be greatly sacrificed. On the other hand, if the entire length of the received sequence be considered before a decision is made, the complexity of decoder and time consumed would be astronomically high.

A radical innovation, which alleviated this problem was the proposal of the algorithm called "sequential decoding" by Wozencraft [1957].²⁴ The algorithm was subsequently refined by Reiffen¹⁵ and Fano.¹⁶

The basic idea of sequential decoding can be illustrated using the tree diagram which is simply another form of the trellis diagram or the state diagram. The tree structure can be extended indefinitely to the right; however, it repeats itself after k levels due to the recurrent nature of the code. There are 2^k branches stemming from each node for an (n, k) code.

ORIGINAL PAGE 11
OF POOR QUALITY

ORIGINAL PAGE IS
OF POOR QUALITY

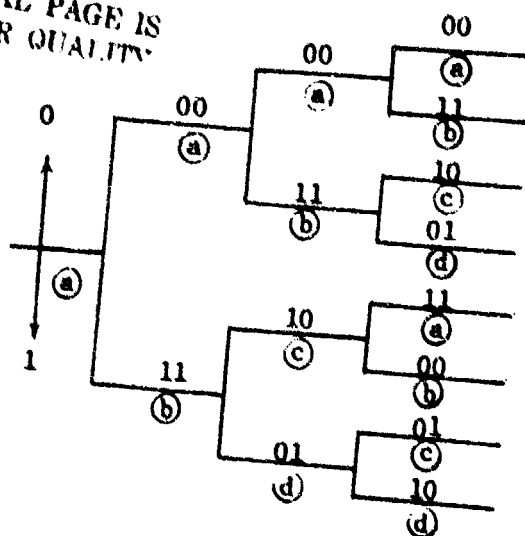


Fig. 2.6 Tree diagram.

With the aid of the tree diagram, encoding is simply a tracing of the tree branches based upon the source sequence and decoding becomes a reversed procedure of retracing the tree branches according to the information contained in the received sequence. Since there are only 2^l branches stemming from each node, only 2^l alternations have to be decided upon at each decoding operation. This is equivalent to truncating the received sequence into the shortest blocks possible and decoding each block algebraically according to the MLJ rule; therefore, a tremendous amount of work can be saved. Intuition tells us immediately that the overall decoding error would be far from optimal due to the truncation of the received sequence to such short blocks. The remedy is to allow the decoder to look back as deeply as circumstances require to retrace other paths if the accumulated errors committed by the present path violated a certain running threshold. During the searchback period, incoming data must

be temporarily held in a buffer storage. If the channel is very noisy for a certain period of time and the searchback is extensive, a tremendous amount of buffer capacity will be required. Unless the buffer capacity is made infinite, there is always a finite possibility of buffer overflow owing to the fact that there is always a finite possibility of having extensive search to the point where overflow occurs. In order to avoid buffer overflow for a certain fixed buffer size, the channel information rate must be limited to a figure much lower than the capacity of the channel; this figure is termed computational cutoff rate R_{comp} . Although the sequential decoder maps the received sequence to the code space according to the MLD rule, the additional error caused by buffer overflow or alternatively, the limiting of channel rate to R_{comp} renders it a suboptimal decoder.

For those who are familiar with optimal control theory, sequential decoding can be viewed as a discrete linear tracking problem and the algorithm is analogous to the method of dynamic programming. The objective is to minimize the Hamming distance between the code sequence and the received sequence. The admissible state trajectories consist of the entire code space and the control policy is the decoded message sequence.

Since searchbacks impose a severe limitation on the performance of the sequential algorithm, a novel approach is to keep a running score or branch metric on each of the 2^{k-1} most likely paths leading to each state. The selection of the desired path is postponed until some time later to avoid a hasty decision which might be regretted later. This is the basic idea of the optimal MLD algorithm developed

by Viterbi.²⁵ If all the possible paths have to be saved and decision making has to be postponed indefinitely, this algorithm would have no advantage at all, because the entire semi-infinite sequence would be treated as one whole block. However, the objective is to find the path that correlates best with the received sequence. It can be shown²⁶ that among the 2^k paths entering each state at each level, only one path, i.e. the path that correlates best with the received sequence up to that stage, can be a possible candidate for the overall maximum likelihood path; the rest of the paths can be discarded. This will leave only 2^k paths, called the survivor paths, to be considered as decoding progresses along the trellis diagram. Furthermore, as mentioned earlier in this section, if a decision is made after $5(k-1)$ stages have progressed, the additional error probability due to truncation will be comparable to the MLD error probability and even less significant if the received sequence is truncated to length T so that $d_T > d_{\text{free}}$. Since the Viterbi algorithm does not involve searchbacks, buffer overflows do not occur and full channel capacity can be utilized. For these reasons, this algorithm is regarded as having truly maximum likelihood and as being optimal for the decoding of convolutional codes. The price paid for this optimality is the complexity of the hardware which grows exponentially with constraint length or linearly with the number of internal states. However, if the constraint length is limited to less than 10 stages, the hardware complexity will not grow beyond realizability. An additional advantage is that Viterbi decoding is performed in real time, on a bit-by-bit basis; therefore, an input buffer is not

required. This is the algorithm intended to be used as part of the hybrid system and receives considerable attention in the next chapter where its burst error correcting capability will be discussed in detail.

Details of the two probabilistic decoding algorithms discussed above can be found from references 15, 16, 25, 24, 27 28, and 29.

2.4 Special Codes--Codes for Burst Errors

So far the discussions about error-correcting capabilities of convolutional codes have been restricted to their applications to DMCs, i.e. channels containing random errors only. The free distance proposed by Costello¹⁸ as a measure of the random-error-correcting capability of a code does not apply directly to its burst-error-correcting capability. Codes suitable for some special burst error patterns are designed at the sacrifice of their random-error-correcting capabilities and are thus not generally optimal.

Burst-error-correcting convolutional codes are divided into types B1 and B2. The definitions can be found in Shu-Lin's An Introduction to Error-correcting Codes,²¹ Chapter 12.

All the burst-error-correcting codes work well only when the channel is segmented with error-free intervals which are called guard spaces. This is why their applications are severely limited. A type B1 code is capable of correcting a burst of length L , i.e. L burst-error correcting code, provided that an error-free guard space of $nk-1$ digits is given. A type B2 code is capable of correcting a burst of length $L = \lambda n$, i.e. L phased-burst-error-correcting code, provided that a clean guard space of $n(k-1)$ digits is given. Since a

burst of length $(\lambda-1)n+1$ can at most affect λ consecutive blocks, a type B2 code with phased-error-correcting capability $L = \lambda n$ can be regarded as a type B1 code with burst-error-correcting capability $L' = (\lambda-1)n+1$. As L becomes large (for fixed n) the error-correcting capability between the two becomes very close. Since the B2 codes, due to the additional restriction, are easier to analyze, and their performances are not very much different from the B1 codes, most previous studies were conducted in this area. Lower bounds on the length of the guard space required for fixed L were established by Wyner and Ash.³⁰ This bound states that for a type B2 (n,k) code, the constraint length k must satisfy the following inequality:

$$k \geq \frac{n + \ell}{n - \ell} \lambda + 1$$

or equivalently,

$$n(k-1) \geq \frac{n + \ell}{n - \ell} (\lambda n)$$

where $n(k-1)$ is the guard space and $L = \lambda n$, the phased-burst-error correcting capability. Therefore, (guard space required) $\geq \frac{n + \ell}{n - \ell} \times$ (burst length). An optimal B2 code is a code that satisfies the above equation with equality. It can be seen that according to the Wyner-Ash bound, the ratio of guard space length to maximum burst length is exactly $\frac{n + \ell}{n - \ell}$ for an optimal B2 burst-error-correcting convolutional code.

The first burst-error correcting convolutional code was introduced by Hagelbarger [1959]³¹; it was later refined by Peterson [1961].³² Iwadare [1968]³³ discovered two classes of type B1 codes which require

much shorter guard space than the corresponding Hagelbarger codes. Type B2 codes were first studied by Wyner and Ash [1963].³⁰ They also found some optimal B2 codes; these are $(n, n-1)$ codes for $n = 2, 3$, and 4. Subsequently, Berlekamp [1964]³⁴ formulated a general procedure to construct optimal B2 $(n, n-1)$ codes for any n . Although the burst-error-correcting capabilities of these optimal B2 codes are attractive, it is important to realize that these are codes specially designed for burst error correction; in general their random-error-correcting capabilities are greatly reduced.

Codes for both burst and random errors do exist; for instance, the diffuse code and the Gallager's adaptive decoding scheme; however, the former is a systematic orthogonalizable code and the latter is applicable only to systematic orthogonalizable codes. None of them is optimal in burst and/or random error correction.

Special techniques such as interlacing or concatenation are not desirable because they either achieve a burst-error-correcting capability at the sacrifice of random-error-capability or introduce rate loss by using two codes, one inside the other. Furthermore, it is impossible to make their performances approach optimum for both random and burst errors.

2.5 Conclusion

Discussions in this chapter can be summarized as follows:

(1) In general, convolutional codes are better random-error correctors than block codes.

(2) The best decoding algorithm for convolutional codes is the optimal maximum likelihood Viterbi decoder.

(3) The performance of the Viterbi decoder for channels with burst errors or with both random and burst errors is largely unknown. An original study is deemed necessary and will be the main efforts of the next chapter.

(4) Good convolutional codes are those with large free distance; therefore, they are not likely to be systematic codes.

(5) Algebraic decoders truncate the received sequences into short blocks; thus much of the random error correcting capabilities of the convolutional codes are not used. However, they provide better control over burst errors. Since all the known algebraic decoding algorithms are designed for systematic codes, an algebraic decoder for nonsystematic codes must be developed. This will be the topic of Chapter IV.

(6) Special codes designed for strictly burst channels do exist; however, they are not suitable for random errors.

(7) Special techniques for correcting combinations of burst errors and random errors are also available, but they either introduce rate loss--for instance, concatenation--or reduce the random error capabilities of the codes, for instance interlacing. Therefore, these techniques will be excluded from further consideration other than for possible additional protection over the hybrid systems.

CHAPTER III

PERFORMANCE OF THE VITERBI DECODING ALGORITHM FOR COMPOUND CHANNELS

3.1 Introduction

As mentioned in the last chapter, the random-error-correcting capability of the Viterbi decoding algorithm is largely unknown. Existing work in this area is quite limited. Viterbi decoders are generally very good for random error channels but are not as efficient for burst errors;³⁵ such error patterns result in bursts of errors from the Viterbi decoder of approximately 10 to 20 bits, depending on the decoder constraint length and the encoder used.³⁶

More recent studies have reported the performance through analytical statements and computer simulations of short constraint length convolutional codes with binary phase shift keying (BPSK) modulation and Viterbi decoding for communication channels with classical Rician Fading.³⁶ Analytical predictions of performance of convolutional codes have also been derived for bursty channels;³⁷ however, interleaving was used prior to Viterbi decoding. This is more like a study of the potential burst-correcting capability of the codes rather than the performance of the decoding algorithm. Interleaving tends to improve the burst-correcting capability at the sacrifice of the random-error-correcting capability for a particular code. As mentioned earlier, physical channels in nature do consist of both types of errors. It would make more sense if methods of evaluating

the performance of the Viterbi decoders over channels with various degree of memory could be constructed.

Normally, compound channels are difficult to analyze although performance may be characterized through statistical simulation of various encoders and decoders for compound channels with various ratios of burst to random error statistics.

The following sections present the method and the results of a computer simulation investigation of the performance of Viterbi decoders when receiving data corrupted with burst and random errors on the same channel, the compound channel.

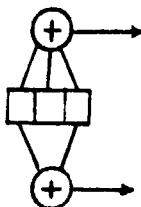
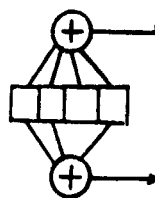
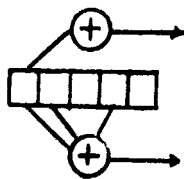
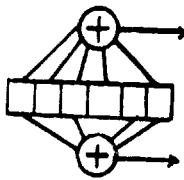
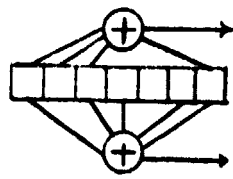
3.2 Method of Study and Some Preliminary Findings

Five encoders were modeled (see Table 3.1) ranging from a constraint length of 3 to 7 and free distance of 5 to 10. Two decoder constraint lengths were utilized and overall channel statistics of 10^{-3} , 10^{-2} , 10^{-1} bit error rates were simulated. The burst errors were inserted at random within the random error stream and burst lengths of 2, 3, 4, and 5 or more bits were randomly distributed through the data with appropriately varying statistics.

A comprehensive set of computer simulations for Viterbi decoders with random and bursty error inputs was generated. The complex channels were simulated using a random number generator as were the source sequences. Although convolutional codes are linear group codes and their properties will be invariant if the all zero sequence is used, for aesthetics and for detecting any nonlinearities which might occur in the hybrid decoding algorithm, the use of randomly generated source sequences seems more preferable.

TABLE 3.1

TYPES OF DECODERS INVESTIGATED

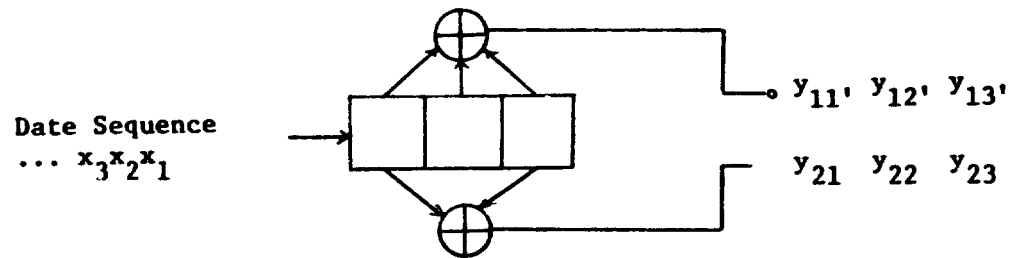
<u>DECODER TYPE</u>	<u>K</u>	<u>V</u>	<u>HOOKUP VECTOR</u>	
A	3	2	00007	
			00005	
B	4	2	00017	
			00015	
C	5	2	00020	
			00032	
D	6	2	00075	
			00053	
E	7	2	00171	
			00133	

The computer simulation of the Viterbi decoding algorithm can best be explained with an example, using hand calculations. In the following illustrations, the rate $\frac{1}{2}$, $K = 3$ code discussed in Sec. 2.3 will be considered again. This encoder is rather simple in form and short in constraint length (hence limited in error correction capability) but allows a demonstration of the type of error output sequences which occur from decoders using the Viterbi decoding algorithm. The encoder is a constraint length 3, rate $\frac{1}{2}$ encoder shown in Fig. 3.1. A typical input data sequence and coded output sequence is shown in Fig. 3.2. This code has a d_{free} of 5^{38} and is typically capable of correcting two bit error patterns in a received sequence 6 bits long.

3.2.1 Decoder Operation:

A periodic section of the trellis diagram for this code is shown in Fig. 3.3. This code has 4 states ($2^{k-1} = 2^{3-1} = 2^2 = 4$), these states with the corresponding outputs and transactions constitute the periodic section of the trellis diagram.

On a BSC, errors which transform a channel code symbol 0 to 1 or 1 to 0 are assumed to occur independently from symbol to symbol with probability p . If all input (message) sequences are equally likely, the decoder which minimizes the overall error probability for any code, block or convolutional, is one which examines the error-corrupted received sequence $y_1, y_2, \dots, y_j, \dots$ and chooses the data sequence corresponding to the transmitted code sequence $x_1, x_2, \dots, x_j, \dots$ which is closest to the received sequence in the sense of Hamming distance; that is, the transmitted sequence which differs



Data Sequence Input $x_1, x_2, x_3, \dots$

Data Sequence Output $y_{11}y_{21}, y_{12}y_{22}, y_{13}y_{23}, \dots$

$$\text{Where } y_{1i} = x_i + x_{i-1} + y_{i-2}$$

$$y_{2i} = x_i + x_{i-2}$$

Figure 3.1 Convolutional encoder (k=3)

Input Data Sequence	0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1...
Coded Output Sequence	00 00 11 10 00 01 01 00 10 00 10 11 11 01 01 00 ...

Figure 3.2 Input and output data sequences

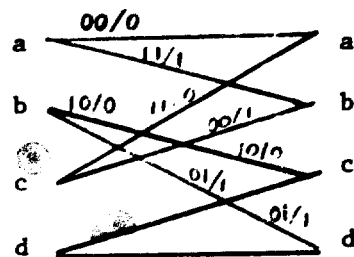


Fig. 3.3 Periodic portion of trellis diagram.

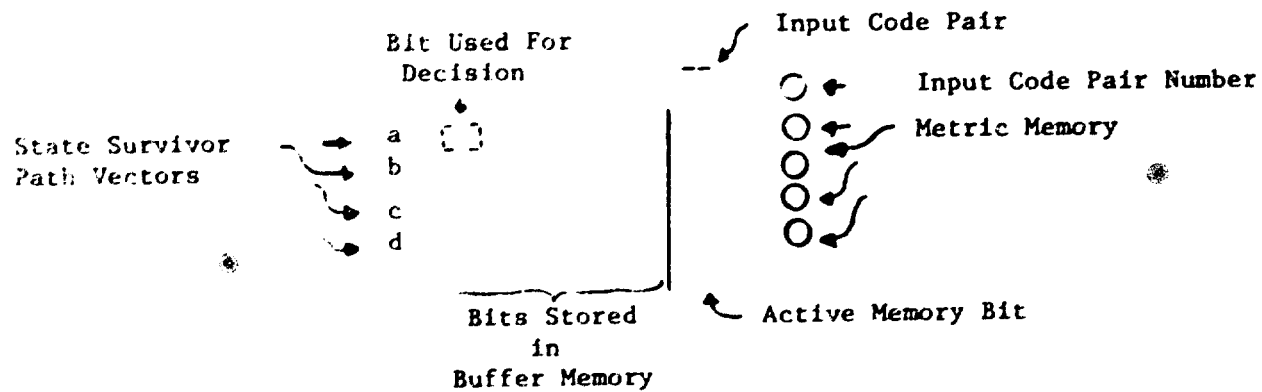


Fig. 3.4 Illustrative diagram of decoder operation elements.

from the received sequence in the minimum number of symbols.

Referring first to the tree diagram of the coder, this implies that the path in the tree whose code sequence differs is the minimum number of symbols from the received sequence should be chosen. However, recognizing that the transmitted code branches merge continually, the choice may be equally limited to the possible paths in the trellis diagram. Examination of this diagram indicates that it is unnecessary to consider the entire received sequence (which conceivably could be thousands or millions of symbols in length) at the same time, deciding upon the most likely (minimum distance) transmitted sequence. In particular, immediately after the third branch, which of the two paths leading to node or state a is more likely to have been sent can be determined. For example, if 010001 is received, it is clear that this is at distance 2 from 000000 while it is at distance 3 from 111011; consequently, the lower path into node a may be excluded. For no matter what the subsequent received symbols will be, they will affect the distances only over subsequent branches after these two paths have merged and in exactly the same way. The same can be said for pairs of paths merging at the other three nodes after the third branch. The minimum distance path of the two paths merging at a given node will be referred to as the "survivor". Thus it is necessary only to remember which was the minimum distance path from the received sequence (or survivor) at each node, as well as the value of that minimum distance. This is necessary because at the next node level, the two branches merging at each node level must be compared to determine which were survivors at the previous level for different

nodes. For example, the comparison at node a after the fourth branch is between the survivors of comparisons at nodes a and c after the third branch. Thus, if the received sequence over the first four branches is 01000111, the survivor at the third node level for node a is 000000 with distance 2 and at node c it is 110101 also with distance 2. In going from the third node level to the fourth, the received sequence agrees precisely with the survivor from c but has distance 2 from the survivor from a. Hence the survivor at node a of the fourth level is the data sequence 1100 which produced the code sequence 11010111 which is at (minimum) distance 2 from the received sequence.

In this way, it is possible to proceed through the received sequence and at each step for each state preserve one surviving path and its distance from the received sequence, which is more generally called metric. The only difficulty which may arise is the possibility that in a given comparison between merging paths, the distances or metrics may be identical. Then one may simply flip a coin as is done for block codewords at equal distances from the received sequence. For even if both of the equally valid contenders are preserved, further received symbols would affect both metrics in exactly the same way and thus not further influence the choice.

The internal operations of the decoder have been shown in the following examples. At the bottom of each of the examples the equivalent input and output error sequences have been shown. The arrows indicate the output digit the decoder will select for a buffer memory of 9 digits. Fig. 3.4 illustrates the make up of the block of

operations and memories used in each decision step of the decoder.

For the $k = 3$ rate = $1/2$ encoder there are 4 survivor paths and 4 metrics to remember. For the $k = 7$ rate = $1/2$ encoder there are 64 survivor paths and 64 metrics to remember. Thus, simulation of the $k = 7$ rate = $1/2$ encoder will be considerably more involved than it would be for a $k = 3$ rate = $1/2$ encoder.

3.2.2 Preliminary Findings

Immediate results from this short constraint length code can be summarized as follows:

1. Viterbi decoder constraint lengths DCL (decoder buffer memory storage capability in bits) is an important parameter relating to performance. In general, the ratio of error clusters to decoder constraint length is directly proportional to unsatisfactory performance. In the example presented in Section 3.2.1, it is clearly evident that although a buffer memory of 6 performs well for random errors, where it is very unlikely for 2 or more errors to occur within a space of a few bits of data, it produces a very poor performance for bursty error situations such as 2, 3, 4, or 5 errors confined within a short span of data. In fact, increasing the DCL to 9 results in much better performance. This is to be expected, but the questions are what is the relationship for longer constraint length codes and where is the point of limiting returns?
2. Short spans of errors often produce more errors than a

slightly longer span of errors; consider the 3 error case versus the 4 error case in the example.

3. If a DCL is too short, the error pattern of the output stream is very different from the error pattern of input data stream. In particular the delay in the decoder allows errors in the output stream to occur in bits well before the actual errors which occurred in the original data stream. This phenomena can be very bothersome to the synchronizer pattern detector of a communication system and can result in synch lock loss which results in large data losses.³⁹
4. Studies to date do not address the above questions.
5. Once a Viterbi decoder is forced to restart decoding due to a situation such as synch loss which creates a large state metric error within the decoder, the first 3 to 4 constraint lengths of data will be unreliable due to the unknown encoder starting state.³⁵
6. Random error studies have shown that errors from a Viterbi decoder do tend to occur in short bursts of 10 to 20 bits²⁵ which could easily create either false synch loss or uncorrectable error patterns in the data identification words of a communication system.³⁹
7. Decoder synchronization must take place if the decoder loses its reference as to which incoming pair of bits constitutes a code sequence pair. This synchronization process usually takes from four to five constraint lengths

Decoding With 3 Errors Using Buffer Memory 9 and 6

DATA INPUT ... 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 0
 CODE SEQUENCE.. 00 11 10 00 01 01 00 10 00 10 11 11 01 01 00 10 11 11 01 01 00 10 11 00 00
 RECEIVED SEQUENCE. 00 11 10 00 01 01 00 11 01 11 11 11 01 01 00 10 11 11 01 01 00 10 11 00 00
 Starting with Last Correct Sequence Received

																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					</
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

DATA INPUT SEQUENCE ... 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0
 EQUIVALENT ERROR SEQUENCE . x x x
 DATA OUTPUT SEQUENCE 0 1 0 1 1 0 1 1 0 0 0 1
 EQUIVALENT ERROR SEQUENCE x x
 USING BUFFER MEMORY 6:
 DATA OUTPUT SEQUENCE ... 0 0 0 1 1 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0
 x x x

Figure 3.5

ORIGINAL PAGE IS
 OF POOR QUALITY

Lets Decode One With 4 errors Using Buffer Memory 9 and 6 (9 indicated)

DATE INPUT: 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 0

CODE SEQUENCE: 00 11 10 00 01 01 00 10 00 10 11 11 01 01 00 10 11 11 01 01 00 10 11 00 00

RECEIVED SEQUENCE: 00 11 10 00 01 01 00 11 11 00 11 11 01 01 00 10 11 00 00

We Start With Last

Correct Sequence Received

a	00 01 01 10	0 0 00 00 01 10	11 07	0 0 01 01 10 10	11 08	0 0 10 11 01 00	00 09	1 01 10 10 00	11 10
b	00 01 01 10	1 0 00 00 10 11 00	0 0 00 00 11 00	0 0 01 01 10 11	0 0 00 10 11 01	0 0 10 11 01 10	1 01 10 10 00	1 01 10 10 00	0 0 00 00
c	00 00 00 11	0 0 00 10 11 01	0 0 01 01 10 11	1 0 01 01 10 11	0 0 10 11 01 11	0 0 10 11 01 11	0 0 10 11 01 11	1 01 10 11 01	0 0 00 00
d	00 00 00 11	1 0 00 10 11 01	1 0 01 01 10 11	1 0 01 01 10 11	1 0 01 01 10 11	1 0 01 01 10 11	1 0 01 01 10 11	1 0 01 01 10 11	1 0 01 01 10 11
	0 11 01 10 10	0 0 10 10 00 10	0 0 10 10 00 10	0 0 10 10 00 10	0 0 10 10 00 10	0 0 10 10 00 10	0 0 10 10 00 10	0 0 10 10 00 10	0 0 10 10 00 10
	0 11 01 00 00	1 0 10 10 00 10	1 0 10 10 00 10	1 0 10 10 00 10	1 0 10 10 00 10	1 0 10 10 00 10	1 0 10 10 00 10	1 0 10 10 00 10	1 0 10 10 00 10
	0 11 01 00 01	0 0 10 10 00 11	0 0 10 10 00 11	0 0 10 10 00 11	0 0 10 10 00 11	0 0 10 10 00 11	0 0 10 10 00 11	0 0 10 10 00 11	0 0 10 10 00 11
	0 11 01 00 01	1 0 10 10 00 01	1 0 10 10 00 01	1 0 10 10 00 01	1 0 10 10 00 01	1 0 10 10 00 01	1 0 10 10 00 01	1 0 10 10 00 01	1 0 10 10 00 01
	0 00 01 10 10	0 0 00 11 01 00	0 0 00 11 01 00	0 0 00 11 01 00	0 0 00 11 01 00	0 0 00 11 01 00	0 0 00 11 01 00	0 0 00 11 01 00	0 0 00 11 01 00
	0 00 11 00 00	1 0 00 11 01 00	1 0 00 11 01 00	1 0 00 11 01 00	1 0 00 11 01 00	1 0 00 11 01 00	1 0 00 11 01 00	1 0 00 11 01 00	1 0 00 11 01 00
	0 00 11 00 01	0 0 01 10 00 01	0 0 01 10 00 01	0 0 01 10 00 01	0 0 01 10 00 01	0 0 01 10 00 01	0 0 01 10 00 01	0 0 01 10 00 01	0 0 01 10 00 01
	0 00 11 00 01	1 0 01 10 00 01	1 0 01 10 00 01	1 0 01 10 00 01	1 0 01 10 00 01	1 0 01 10 00 01	1 0 01 10 00 01	1 0 01 10 00 01	1 0 01 10 00 01
	1 01 00 11 00	0 0 10 01 10 10	0 0 10 01 10 10	0 0 10 01 10 10	0 0 10 01 10 10	0 0 10 01 10 10	0 0 10 01 10 10	0 0 10 01 10 10	0 0 10 01 10 10
	1 01 00 11 00	1 0 10 01 10 10	1 0 10 01 10 10	1 0 10 01 10 10	1 0 10 01 10 10	1 0 10 01 10 10	1 0 10 01 10 10	1 0 10 01 10 10	1 0 10 01 10 10
	1 01 00 11 01	0 0 10 01 10 11	0 0 10 01 10 11	0 0 10 01 10 11	0 0 10 01 10 11	0 0 10 01 10 11	0 0 10 01 10 11	0 0 10 01 10 11	0 0 10 01 10 11
	1 01 00 11 01	1 0 10 01 10 11	1 0 10 01 10 11	1 0 10 01 10 11	1 0 10 01 10 11	1 0 10 01 10 11	1 0 10 01 10 11	1 0 10 01 10 11	1 0 10 01 10 11

DATA INPUT SEQUENCE... 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0

EQUIVALENT ERROR SEQ... x x x x

DATA OUTPUT SEQUENCE.. 0 1 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 Note 10 bit delay

EQUIVALENT ERROR SEQUENCE OUTPUT x

USING BUFFER MEMORY 6:

DATA OUTPUT SEQUENCE . 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 Note 7 bit delay

Equivalent Error

Sequence Output

Figure 3.6

ORIGINAL PAGE IS
OF POOR QUALITY

Decoding With 5 Errors Using Buffer Memory 9 and 6

DATA INPUT ... 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 0
 Code Sequence. 00 11 10 00 01 01 00 10 00 10 11 11 01 01 00 10 11 11 01 01 00 10 11 00 00
 Received Seq.. 00 11 10 00 01 01 00 11 01 11 10 11 00 01 00 10 11 11 01 01 00 10 11 00 00

Starting With Last Correct Sequence Received:

	+			00	6	+		x				01	x				x	
a		00	01	01	10		0	11	7	+		0	11	9	+		10	10
b		00	01	01	10		1	0	3	0	01	01	10	10	10	10	0	2
c		00	00	00	11		0	1	2	0	01	01	10	10			1	2
d		00	00	00	11		1	0	1	0	01	01	10	11			1	2
							1	1	1	0	01	01	10	01			1	3

DATA INPUT SEQUENCE ... 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0
 EQUIVALENT ERROR SEQ... x x x x x
 DATA OUTPUT SEQUENCE... 0 1 0 1 1 0 1 1 0 0 0 1 1 0 1 0 0
 EQUIVALENT ERROR SEQ... x x
 USING BUFFER MEMORY 6..
 DATA OUTPUT 0 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0
 x x

Figure 3.7

Decoding With 5 Errors Using Buffer Memory 9 and 6

DATA INPUT ... 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0
 Code Sequence. 00 11 10 00 01 01 00 10 00 10 11 11 01 01 00 10 11 11 01 01 00 10 11 00 00

Received Seq... 00 11 10 00 01 01 00 11 00 11 10 11 10 01 00 10 11 11 01 01 00 10 11 00 00
 Starting with Last Correct Sequence Received:

a	00 01 01 10	00 ⑥	11 ⑦	00 ⑧	11 ⑨	10 ⑩
b	00 01 01 10	0 ②	0 ③	0 ④	0 ⑤	0 ⑥
c	00 00 00 11	1 ②	0 ①	0 ②	0 ③	0 ④
d	00 00 00 11	1 ③	0 ①	0 ②	1 ②	1 ②

01 10 00 10	11 ⑪	10 ⑫	01 ⑬	00 ⑭	10 ⑮
0 11 01 10 00	0 ③	0 ④	0 ⑤	0 ⑥	0 ⑦
0 11 01 01 11	1 ③	1 ④	1 ⑤	1 ⑥	1 ⑦
0 11 01 01 11	1 ③	1 ④	1 ⑤	1 ⑥	1 ⑦

01 11 10 10	11 ⑯	01 ⑰	01 ⑱	00 ⑲	00 ⑳
1 00 01 00 00	0 ③	1 ④	1 ⑤	1 ⑥	1 ⑦
1 00 01 00 01	1 ⑤	0 ⑥	0 ⑦	0 ⑧	0 ⑨
0 11 11 10 11	1 ⑥	0 ⑦	0 ⑧	0 ⑨	0 ⑩

DATA INPUT SEQUENCE... 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0
 EQUIVALENT ERROR SEQ.. x x x x
 DATA OUTPUT SEQUENCE.. 0 1 0 0 1 0 1 0 1 1 1 1
 EQUIVALENT ERROR SEQ.. x x x
 USING BUFFER MEMORY 6.
 DATA OUTPUT SEQUENCE.. 0 0 0 1 1 1 0 1 0 0 1 1 1 0 1
 x x x x x

Figure 3.8

of received signals; of course, during this time the output is highly unreliable data.

3.3 Results and Findings of Computer Simulations

Computer simulations of Viterbi decoders exhibiting input data with both random and bursty error patterns are summarized in this section. The data runs were made for three Nominal Error Probabilities, 10%, 1%, 0.1%, with the burst error percentages composing half the total percentage. Bursts were spread over various ranges with lengths 1, 2, 3, 4, and 5 or more with varying percentages of the total error percentage.

Fig. 3.9 shows a typical print out for a data run with the statistics that were compiled.

Data runs were made for both 2000 word strings and 200 word strings to determine if significant differences might be encountered, thereby implying that a 2000 word string might not be long enough. Two decoder constraint lengths were used to ascertain the effects of this parameter and five encoder constraint lengths were used to determine whether longer encoders achieved better performance than short encoders. The channel error statistics used were rather noisy, varying from error rates of 10^{-1} to 10^{-3} bit error rates. These rates allowed shorter computer runs while the performance characteristics were still indicated.

Table 3.1 illustrates the type of decoders modeled with the type E being comparable to the Viterbi decoder being used with the Space Shuttle Communication System.³⁹

CONSTITUTIONAL CONCERN--VITLMDI JACQUER WITH
KINDLY ABOUT DEBATING. ALSO RAISON EKRJA VLLTON

NO. OF SUBJECTS = 50
NO. OF BITS PER SUBJ = 32
1014 NO. OF BITS = 968

[illegible]

(10) CRACK CALCULATED FOR ACTUAL / HANDED IN THE CHANNEL

140. OF CDR SHAW'S	=	30
140. OF B-1's PER HOUR	=	64
101A NO. OF B-1's	=	1870
101A NO. OF AIRBORN B-1's	=	191
CARRON P-40. PER HUR	=	.701250 I
CARRON P-40. PER HOUR	=	.137213 I

NO. OF ENTRIES IN A ROW	NO. OF TIMES
1	1
2	3
3	3
4	3
5	1
Total	11

ERROR PROB. IN %
(IN TERMS OF FREQ. OF OCCUR.)

[illegible]

בנימין נתניהו (19)

NO. OF CODE	=	30
OF HTS.	=	69
NO. OF HTS	=	108
NO. OF COUNTRY HTS	=	15
COUNTRY PROD. PER DT	=	.701250 %
COUNTRY PROD. PER CTRY	=	.157233 %

NO. OF EXPORTS/ADMN	NO. OF TIMES
0	21
1	3
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	1
21	1
22	1
23	1
24	1
25	1
26	1
27	1
28	1
29	1
30	1
31	1
32	1
33	1
34	1
35	1
36	1
37	1
38	1
39	1
40	1
41	1
42	1
43	1
44	1
45	1
46	1
47	1
48	1
49	1
50	1
51	1
52	1
53	1
54	1
55	1
56	1
57	1
58	1
59	1
60	1
61	1
62	1
63	1
64	1
65	1
66	1
67	1
68	1
69	1
70	1
71	1
72	1
73	1
74	1
75	1
76	1
77	1
78	1
79	1
80	1
81	1
82	1
83	1
84	1
85	1
86	1
87	1
88	1
89	1
90	1
91	1
92	1
93	1
94	1
95	1
96	1
97	1
98	1
99	1
100	1
101	1
102	1
103	1
104	1
105	1
106	1
107	1
108	1
109	1
110	1
111	1
112	1
113	1
114	1
115	1
116	1
117	1
118	1
119	1
120	1
121	1
122	1
123	1
124	1
125	1
126	1
127	1
128	1
129	1
130	1
131	1
132	1
133	1
134	1
135	1
136	1
137	1
138	1
139	1
140	1
141	1
142	1
143	1
144	1
145	1
146	1
147	1
148	1
149	1
150	1
151	1
152	1
153	1
154	1
155	1
156	1
157	1
158	1
159	1
160	1
161	1
162	1
163	1
164	1
165	1
166	1
167	1
168	1
169	1
170	1
171	1
172	1
173	1
174	1
175	1
176	1
177	1
178	1
179	1
180	1

[illegible]

NO. OF CIPHERS 211 RUS NO. OF 1845

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
84

9. WF 1845

٥٠٠

EUROPEAN PROD. IN 2
(1) TERMS OF FRLD. OF OCCUR.)

• 060600
• 060600
• 060600
• 060600
• 157233
• 157233

ERROR PROP. IN %
(IN TERMS OF TOTAL NO. OF BITS)

.000000
 .000000
 .000000
 .000000
 .751250
 .751250

(b) CONFIDENTIAL

no. of books	4075
no. of sets	100
total no. of titles	100
total no. of editions	100
total no. of copies	100
total no. of volumes	100

33
34
35

• 342530
• 342530

102. of LARSEN/2011.

552236
Tulal

.. CF 1125

27
30000

END OF PKNB. IN 3

20.939999
 10.000000
 .000000
 .000000
 .000000
 .000000
 100.000000

NO. of LITHUM, AL : 100

IN. OF 1145

1
2
3
4
5
6

1944

(II. TERMS OF FREQ. OF OCCUR.)

- 31250J
- 00000J
- 75000J
- 05000G
- 000300
- 31250J

(11) ^{ERROR PROP. IN %} TERMS OF TOTAL NO. OF BITS

.312500
 .000000
 .000000
 .000000
 .000000
 .312500

ORIGINAL PAGE IS
OF POOR QUALITY

(2) IMPROVEMENT OF THE CHANNEL DUE TO COILING -- IN TERMS OF ERROR PROP. PER BIT

DECLASSING OF ERON PRG. (WITHOUT COILING - WITH COILING) IS	.400750
DECLASSING OF ERON PRG. (WITH COILING / WITHOUT COILING) IS	.400000

Figure 3.9 (cont.) Typical printout of the simulation program - Viterbi decoder

Table 3.2 lists the parameters varied for each type of decoder and Table 3.3 lists the parameters tabulated for each data run.

Some parameters have been summarized by tables to facilitate discussion and to illustrate the performance of these decoders. In Table 3.4 the overall probability of error with coding is compared to the overall probability of error without coding by ratio. These numbers should be interpreted in the following manner. If the coding has helped the system, then the output error probability should lower the error probability without coding. Thus, in the extremes the ideal ratio is 0.0 and the worst ratio approaches ∞ with a ratio of 1.0 indicating that no help is indicated by coding.

The statistics in Table 3.4 can be somewhat misleading if one does not realize that these are overall statistics which do not reflect the reduction of the number of errors in words (if any). In Table 3.5 a comparison of the reduction in the number of errors in words is presented.

A feeling for the benefits obtained may be gained if one realizes that the output of the Viterbi decoder may be corrected further. For instance, the Space Shuttle Digital Data Link will further decode some special data words, the synch words.³⁹ These words, being encoded in a triple error correcting code word, are capable of correcting all errors in a word of 3 bits or less. Thus, it should be determined whether the Viterbi decoder, even in the face of bursty errors, has reduced significantly the errors per word of 4 or more. Data in Table 3.5 indicate that for channels of 10^{-2} BER or less there is a good reduction of these words of 4 errors or more. The significance

of this lies in the synch-lock system of a communication system. The communication system will lose lock and enter a probation or a search mode if more than 3 errors occur in a synch word. If the search mode is entered, data is lost by the frame and this results in considerable data loss.

The Viterbi decoder is rather unpredictable with respect to decoder size versus burst error performance as may be seen from Table 3.5 for the cases of decoders C, D, and E for 0.1% NEP. These decoders are longer than types A and B and in theory should do better in that they are capable of correcting more errors per word than A or B. However, when these decoders make mistakes, they take longer to re-establish the correct bath.

The purpose of a hybrid decoding system is to add a "quick look" decoder which can spot burst errors and the error free space which comes after. The output of the "quick look" decoder may be used during the error free space to reset the Viterbi to a best guess path which should help to alleviate the disadvantage of the longer coders.

Table 3.6 presents a summary of the error statistics for the 2000 word simulation for nominal channel bit error rates of 10^{-2} and 10^{-3} . The actual channel error rates were 1.5289×10^{-2} BER and $.1078 \times 10^{-2}$ BER. The decoded data output bit error rates are presented as well as the probability of x errors per word occurring for $x = 0, 1, 2, 3, 4$, and 5 or more errors per word.

Conclusions to be drawn from these simulation runs are summarized below.

1. The 2000 word simulations for BER of 10^{-1} and 10^{-2} seem

to be long enough data runs for accurate results. The BER of 10^{-3} seems to be long enough with perhaps one or two cases that would be more reliable with longer data runs.

2. The longer DCL decoders for the type C, D and E decoders are necessary for good performance by the decoders. This is reasonable when one considers a normal DCL of 4 to 5 times the encoder constraint length.
3. Error rate improvements of roughly an order of magnitude were typical for the 10^{-3} BER longer DCL decoders.
4. The probability of words occurring in the decoded data stream that contained more than 3 errors (hence rendering a synch word unusable if the errors occurred in a synch word) were reduced significantly for the type E decoder using a DCL of 35 and BER of 10^{-3} . (Probability of four errors or more before decoding was .6000% versus .050% after decoding.)
5. The probability of words occurring in the decoded data stream that contained more than 2 errors (hence rendering a synch word incorrect and creating a possible loss of a frame of data due to false synch loss) was reduced from 1.000 % to .050% after decoding for the type E. DCL of 35 decoder.
6. Longer data runs are necessary for the type E decoder with DCL of 35 to ascertain that the results are not unique to this run.

7. For the same overall channel error rate, the relative order of magnitude of degradation varies from 10 to 100 when the channel contains both random and burst errors as compared to the random only situation, Fig. 3.10-3.14.

TABLE 3.2

PARAMETERS VARIED FOR EACH TYPE DECODER

No. of Source Words: 2000, 200

Total Nominal Error Probability
in Simulation: 10%, 1%, 0.1%

Decoder Constraint Length (DCL):

<u>TYPE DECODER</u>	<u>DCL VARIATIONS</u>
A	9, 15
B	12, 20
C	15, 25
D	18, 30
E	21, 35

TABLE 3.3

PARAMETERS TABULATED FOR EACH TYPE DECODER

- A. Error Statistics Actually Occurring in Error Vector During Simulation
 - 1. Number of Errors in a row: 1, 2, 3, 4, 5 or more
 - 2. Error Probability in % in terms of frequency of occurrence
 - 3. Error Probability in % in terms of total number of bits
 - 4. Number of times a particular number of errors in a row occurred
- B. Error Statistics as They Occur in the Received Sequence Before Decoding
 - 1. Same as A1 above
 - 2. Same as A2 above
 - 3. Same as A3 above
 - 4. Same as A4 above
 - 5. Number of errors per word 1, 2, 3, 4, 5 or more
 - 6. Number of times a particular number of errors in a word occurred
 - 7. Error probability of item 6 in percent
- C. Error Statistics as They Occur in the Decoded Data Sequence
Same Items 1 through 7 for B above.
- D. Improvement of the Channel Due to Coding - in Terms of Error Probability Per Bit
 - 1. Ratio of Error Prob. (with coding/without coding)
[Note ideally the ratio of item D1 is 0.0]
 - 2. Decreasing of Error Prob. (without coding/with coding)

TABLE 3.4

OVERALL CHANNEL IMPROVEMENT RATIO VERSUS TYPE DECODER, DCL, NEP, NSW

DECODER TYPE	DCL	NOMINAL ERROR PROBABILITY (NEP)	NO. OF SOURCE WORDS (NSW)	CHANNEL IMPROVEMENT (PER BIT BASIS)	$\frac{(\text{PROB. OF ERROR WITH CODING})}{(\text{PROB. OF ERROR WITHOUT CODING})}$
A	9	10%	$\frac{2000}{200}$	$\frac{.840089}{.849218}$	
A	9	1%	$\frac{2000}{200}$	$\frac{.448648}{.351064}$	
A	9	0.1%	$\frac{2000}{200}$	$\frac{.362319}{.300000}$	
A	15	10%	$\frac{2000}{200}$	$\frac{.799468}{.826021}$	
A	15	1%	$\frac{2000}{200}$	$\frac{.277976}{.308511}$	
A	15	0.1%	$\frac{2000}{200}$	$\frac{.231884}{.400000}$	
B	12	10%	$\frac{2000}{200}$	$\frac{.900843}{.884518}$	
B	12	1%	$\frac{2000}{200}$	$\frac{.345427}{.404255}$	
B	12	0.1%	$\frac{2000}{200}$	$\frac{.231884}{.500000}$	
B	20	10%	$\frac{2000}{200}$	$\frac{.814191}{.835098}$	

ORIGINAL PAGE IS
NOT AVAILABLE

TABLE 3.4
(Continued)
OVERALL CHANNEL IMPROVEMENT RATIO VERSUS TYPE DECODER, DCL, NEP, NSW

<u>DECODER TYPE</u>	<u>DCL</u>	<u>NOMINAL ERROR PROBABILITY (NEP)</u>	<u>NO. OF SOURCE WORDS (NSW)</u>	<u>CHANNEL IMPROVEMENT (PER BIT BASIS)</u>	<u>(PROB. OF ERROR WITH CODING) (PROB. OF ERROR WITH CODING)</u>
B	20	1%	$\frac{2000}{200}$	$\frac{.239142}{.329787}$	
B	20	0.1%	$\frac{2000}{200}$	$\frac{.188406}{.500000}$	
C	15	10%	$\frac{2000}{200}$	$\frac{.941907}{.9238533}$	
C	15	1%	$\frac{2000}{200}$	$\frac{.603986}{.478723}$	
C	15	0.1%	$\frac{2000}{200}$	$\frac{.666667}{.700000}$	
C	25	10%	$\frac{2000}{200}$	$\frac{.940754}{.937973}$	
C	25	1%	$\frac{2000}{200}$	$\frac{.599898}{.478723}$	
C	25	0.1%	$\frac{2000}{200}$	$\frac{.666667}{.700000}$	

ORIGINAL PAGE IS
OF POOR QUALITY

TABLE 3.4
(Continued)
OVERALL CHANNEL IMPROVEMENT RATIO VERSUS TYPE DECODER, DCL, NEP, NSW

DECODER TYPE	DCL	NOMINAL ERROR PROBABILITY (NEP)	NO. OF SOURCE WORDS (NSW)	CHANNEL IMPROVEMENT (PER BIT BASIS)	(PROB. OF ERROR WITH CODING) (PROB. OF ERROR WITHOUT CODING)
D	18	10%	$\frac{2000}{200}$	$\frac{1.441064}{1.345436}$	
D	18	1%	$\frac{2000}{200}$	$\frac{.812468}{.702128}$	
D	18	0.1%	$\frac{2000}{200}$	$\frac{.768116}{.500000}$	
D	30	10%	$\frac{2000}{200}$	$\frac{1.223415}{1.222390}$	
D	30	1%	$\frac{2000}{200}$	$\frac{.15846}{.170213}$	
D	30	0.1%	$\frac{2000}{200}$	$\frac{.00000}{.00000}$	
E	21	10%	$\frac{2000}{200}$	$\frac{1.705632}{1.795260}$	
E	21	1%	$\frac{2000}{200}$	$\frac{.743996}{.712766}$	

TABLE 3.4
(CONTINUED)
OVERALL CHANNEL IMPROVEMENT RATIO VERSUS TYPE DECODER, DCL, NEP, NSW

DECODER TYPE	DCL	NOMINAL ERROR PROBABILITY (NEP)	NO. OR SOURCE WORDS (NSW)	CHANNEL IMPROVEMENT (PER BIT BASIS)	$\frac{\text{(PROB. OF ERROR WITH CODING)}}{\text{(PROB. OF ERROR WITHOUT CODING)}}$
E	21	0.1%	$\frac{2000}{200}$	$\frac{.855072}{1.100000}$	
E	35	10%	$\frac{2000}{200}$	$\frac{1.452056}{1.654049}$	
E	35	1%	$\frac{2000}{200}$	$\frac{.106285}{.106383}$	
E	35	0.1%	$\frac{2000}{200}$	$\frac{.086951}{.500000}$	

TABLE 3.5

WORD ERROR RATE IMPROVEMENT VERSUS TYPE DECODER - NEP = 10% - 2000 WORD SIMULATION

ERRORS/WORD	NO. OF WORDS RECEIVED IN ERROR	DECODER TYPE: DCL:	NO. OF WORDS IN ERROR AFTER DECODING									
			A	B		C		D		E		
			<u>9</u>	<u>15</u>	<u>12</u>	<u>20</u>	<u>15</u>	<u>25</u>	<u>18</u>	<u>30</u>	<u>21</u>	<u>35</u>
0	1		92	163	135	318	151	151	69	203	71	232
1	14		191	243	138	190	116	119	52	82	44	67
2	21		230	235	213	228	174	168	70	130	68	88
3	35		274	241	233	232	187	192	116	110	86	113
4	57		277	215	241	224	257	256	126	190	103	124
5 or more	1872		936	903	1040	908	1115	1114	1567	1285	1628	1376

WORD ERROR RATE IMPROVEMENT VERSUS TYPE DECODER - NEP = 1% - 2000 WORD SIMULATION

0	1117	1696	1792	1773	1858	1735	1735	1694	1956	1729	1964
1	397	213	161	150	80	123	123	114	11	94	13
2	210	64	31	56	47	30	31	65	11	75	7
3	129	13	5	12	6	51	53	54	4	40	6
4	61	11	3	6	4	54	51	29	4	13	4
5 or more	86	3	2	3	5	7	7	44	14	49	6

WORD ERROR RATE IMPROVEMENT VERSUS TYPE DECODER - NEP = 0.1% - 2000 WORD SIMULATION

0	1933	1977	1984	1985	1988	1979	1979	1978	2000	1973	1999
1	35	21	16	14	11	9	9	9	0	11	0
2	12	2	0	1	1	5	5	7	0	9	0
3	8	0	0	0	0	1	1	1	0	2	0
4	5	0	0	0	0	6	6	1	0	2	0
5 or more	7	0	0	0	0	0	0	4	0	3	1

TABLE 3.6

ERROR STATISTICS FOR 2000 WORD SIMULATION FOR 1% AND 0.1% ERROR RATES

DECODER TYPE	DCL	NEP	ACTUAL ERROR PROB. OF RECEIVED SEQUENCE	ERROR PROB. OF DECODED SEQ.	PROB. OF VARIOUS ERRORS PER DECODED WORD IN PERCENT					
					0	1	2	3	4	5 or more
A	9	1%	1.5289%	.68593%	84.7999	10.6500	3.2000	.6500	.5500	.1500
B	12	1%	1.5289%	.5281 %	88.6499	7.5000	2.8000	.6000	.3000	.1500
C	15	1%	1.5289%	.9234 %	86.7499	6.1500	1.500	2.5500	2.7000	.3500
D	18	1%	1.5289%	1.2421 %	84.6999	5.7000	3.2500	2.7000	1.4500	2.2000
E	21	1%	1.5289%	1.1375 %	86.4499	4.7000	3.7500	2.0000	.6500	2.4500
A	9	.1%	.1078%	.0390 %	98.8499	1.0500	.1000	.0000	.0000	.0000
B	12	.1%	.1078%	.0250 %	99.2499	.7000	.0500	.0000	.0000	.0000
C	15	.1%	.1078%	.07187 %	98.9499	.4500	.2500	.0500	.3000	.0000
D	18	.1%	.1078%	.082812%	98.8999	.4500	.3500	.0500	.0500	.2000
E	21	.1%	.1078%	.09218 %	98.6499	.5500	.4500	.1000	.1000	.1500
A	15	1%	1.5289%	.4250 %	89.5999	8.0500	1.8500	.2500	.1500	.1000
B	20	1%	1.289%	.3656 %	92.9000	4.0000	2.3500	.3000	.2000	.2500
C	25	1%	1.5289%	.9171 %	86.7499	6.1500	1.5500	2.6500	2.5500	.3500
D	30	1%	1.5289%	.2421 %	97.7999	.5500	.5500	.2000	.2000	.7000
E	35	1%	1.5289%	.1625 %	98.1999	.6500	.3500	.3000	.2000	.3000
A	15	.1%	.1078%	.02509 %	99.2000	.8000	.0000	.0000	.0000	.0000
B	20	.1%	.1078%	.0203 %	99.3999	.5500	.0500	.0000	.0000	.0000
C	25	.1%	.1078%	.0718 %	98.9499	.4500	.2500	.0500	.3000	.0000
D	30	.1%	.1078%	.00008 %	100.0000	.0000	.0000	.0000	.0000	.0000
E	35	.1%	.1078%	.009375%	99.9500	.0000	.0000	.0000	.0000	.0500

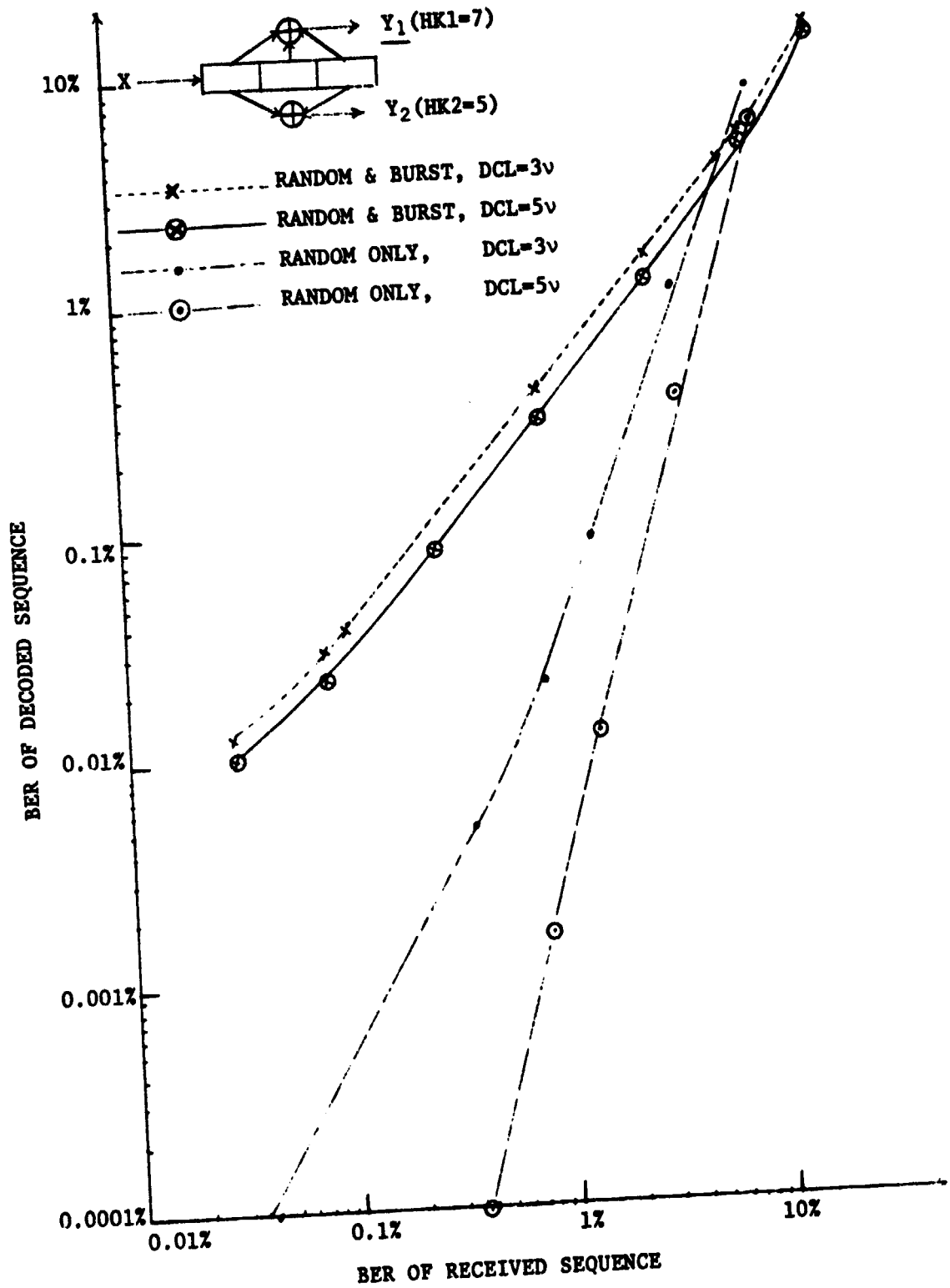


Figure 3.10

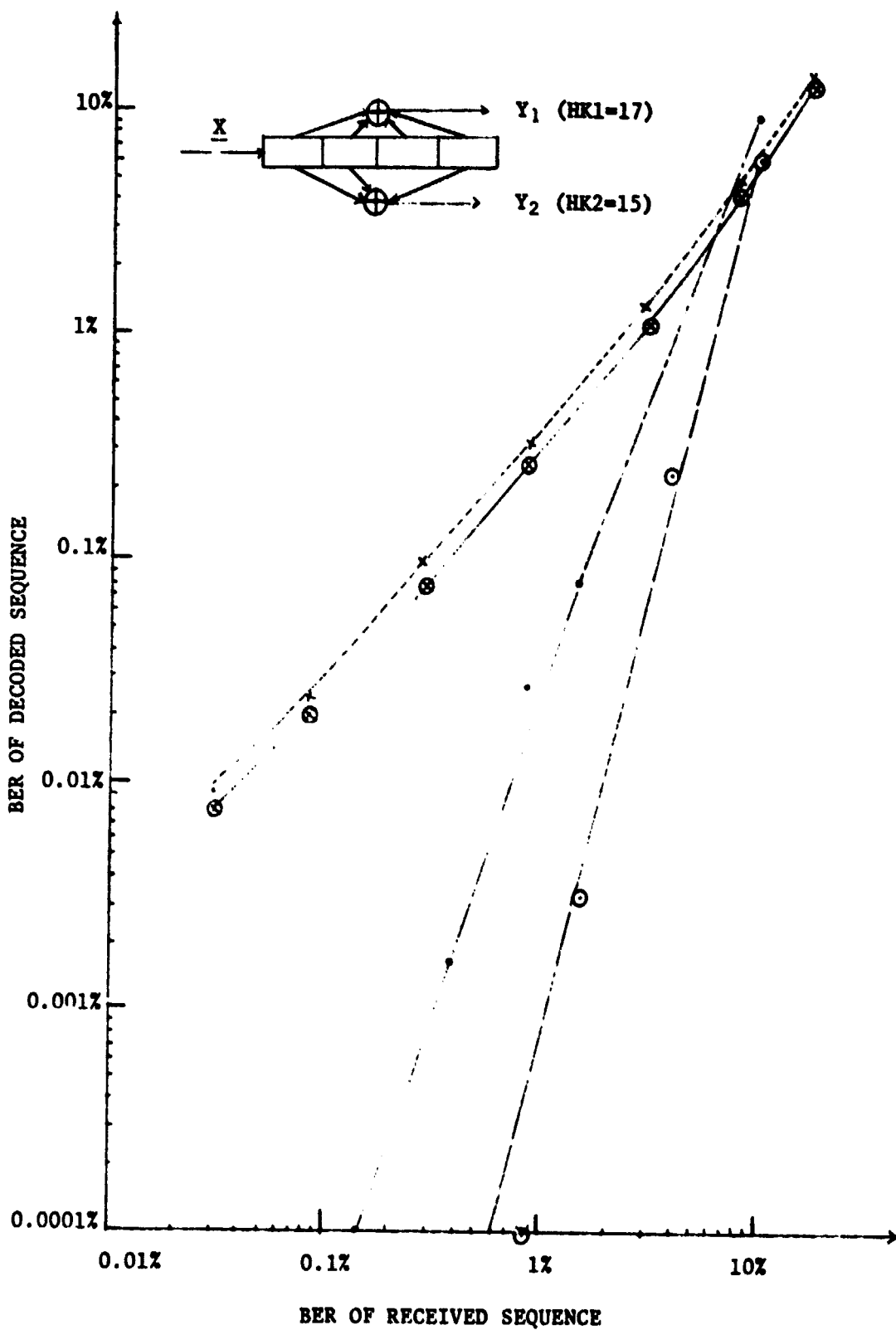


Figure 3.11

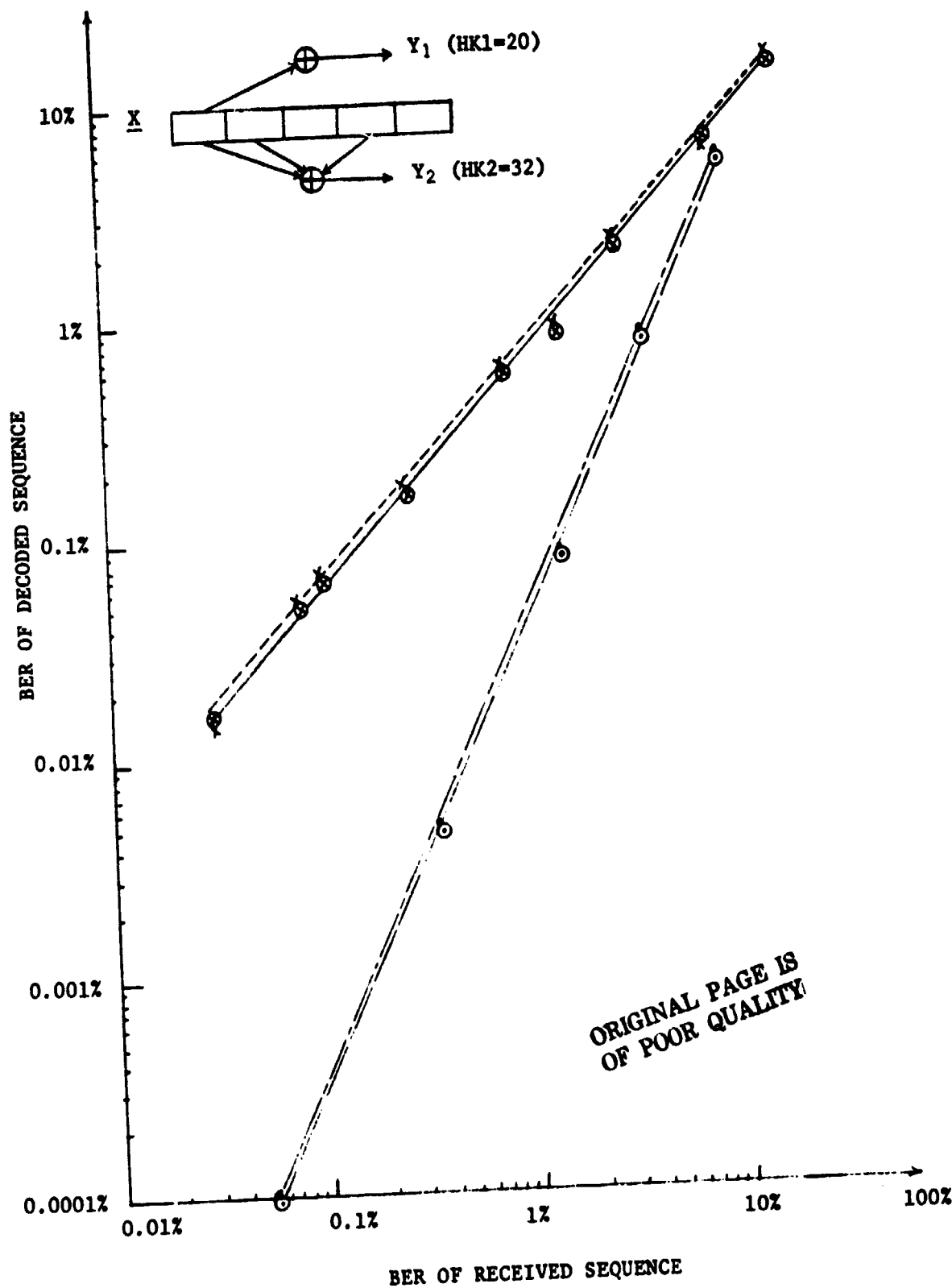


Figure 3.12

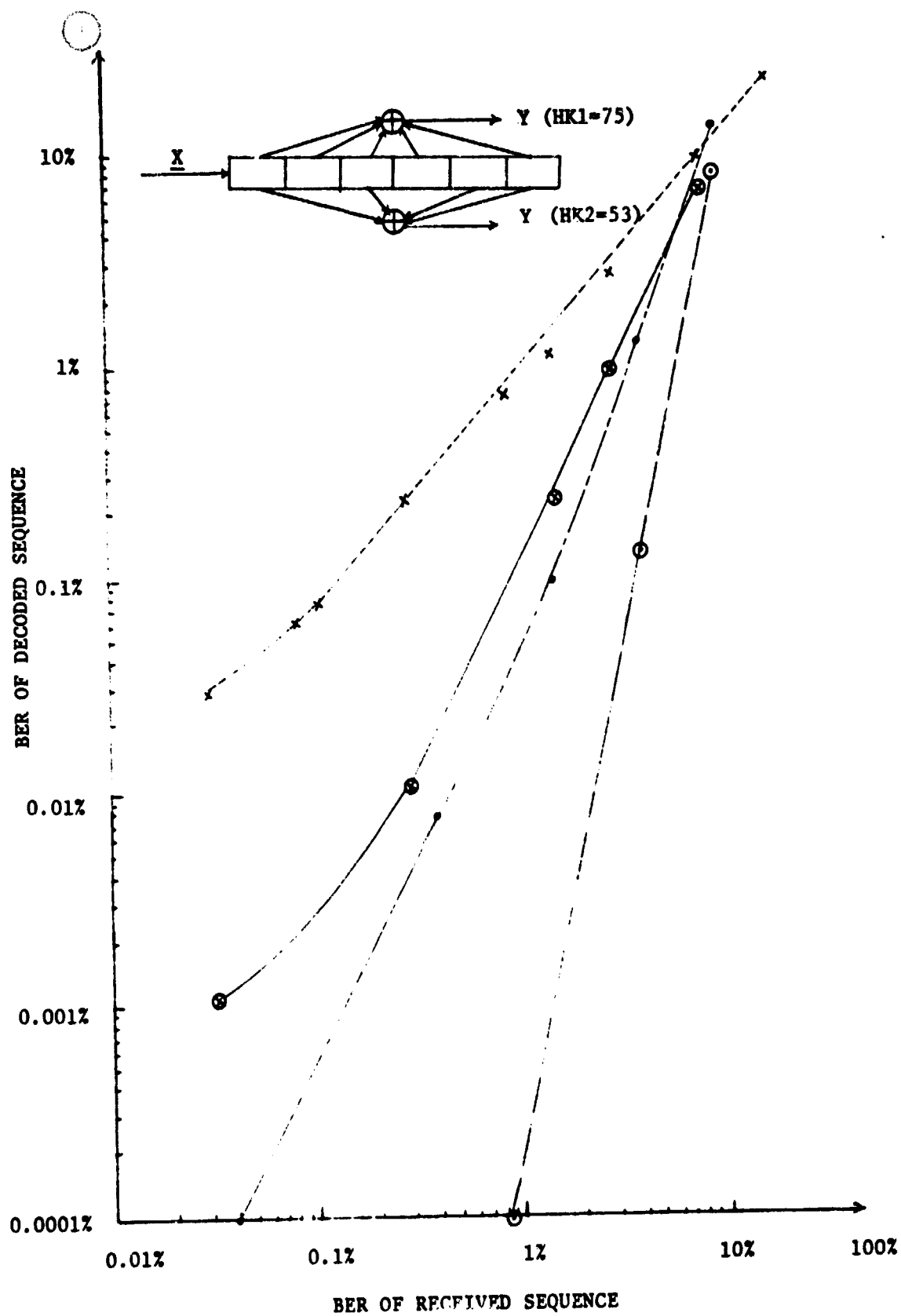


Figure 3.13

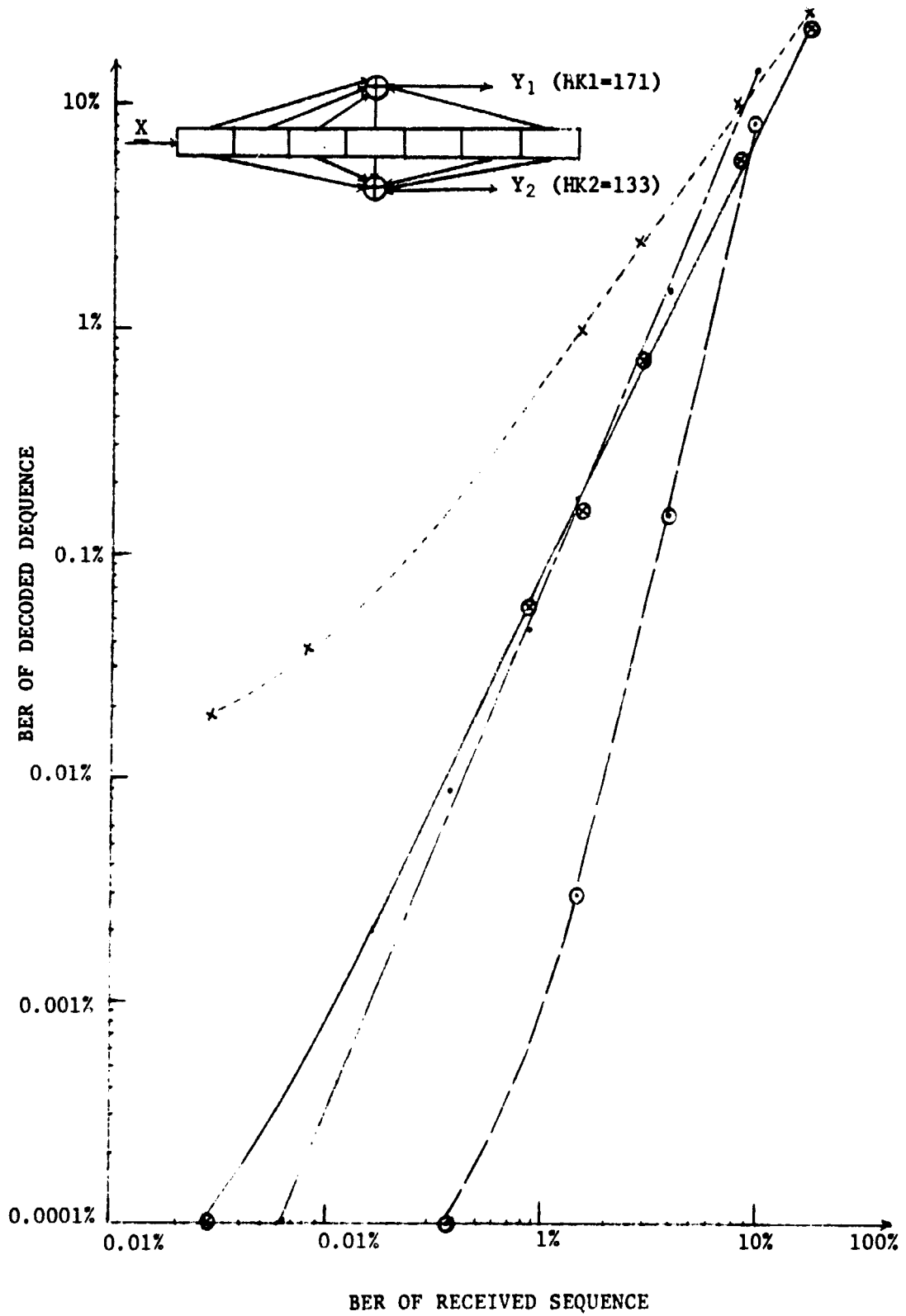


Figure 3.14

CHAPTER IV

ALGEBRAIC DECODER FOR NONSYSTEMATIC CODES

4.1 Introduction

As mentioned in Chapter II, it is highly desirable that decoding algorithms for nonsystematic convolutional codes be developed. Since parity check matrices are difficult to form for this type of codes, an alternate approach must be taken. The method discussed in the following sections truncates the semi-infinite generator matrix G_∞ of a code to a submatrix G_s , called the subgenerator of the corresponding code, such that its inverse, i.e. G_s^{-1} , exists. Thus, encoding is simply the postmultiplication of the message sequence by G_s and decoding, the postmultiplication of the received sequence by G_s^{-1} , i.e. $Y_s = X_s G_s$ and $X_s = Y_s G_s^{-1}$ where X_s and Y_s are the message sequence and the received sequence truncated to segments of length equal to the order of G_s . Obviously, this decoding method works well only when the channel is quiet; it does not provide any error protection for the messages sent.

However, due to the structural properties of the generator matrix G_∞ , the order of the subgenerator G_s can be made an integral multiple of both l and n . This means that if only one block of the message and the received sequence is replaced and multiplied, together with several previous blocks, by their respective multipliers, G_s and G_s^{-1} , at each encoding and decoding operation, multiple parity checks can be provided. This is the basic idea for

the decoding of nonsystematic convolutional codes.

As shown in Chapter II the basic building blocks of the semi-infinite generator matrix of a systematic convolutional code are the $\Gamma + 1$ submatrices, $G_0, G_1, \dots, G_\Gamma$, where each G_i is an $\ell \times n$ matrix. This is still true for a nonsystematic code except that these G_i 's do not contain an identity matrix or a zero matrix of order ℓ . Obviously, the minimal requirement of the subgenerator G_s is that it must be a nonsingular matrix of order δn or a nonsquare matrix of full rank containing all the $\Gamma + 1$ submatrices, where δ is an integral multiple of ℓ . Additional requirements are derived from the consideration of the encoding and decoding of the first block. Boyd²⁰ has been able to show that $\delta \geq \frac{v}{n-\ell}$ and v nonzero rows must be prefixed to G_∞ prior to truncation. The conditions for the existence of G_s can be summarized as follows:²⁰

1. G_s must have at least δn columns where $\delta \geq \frac{v}{n-\ell}$.
2. $\text{rank } \{G_0\}$ must be ℓ .
3. μ must equal v .

μ is the number of the memory elements required in the minimal realization of the encoder and v , in the obvious realization of the encoder. The concept of obvious realization and minimal realization of an encoder will be illustrated in the next section with an example. It is of interest to note that $v = k - \ell = \Gamma \ell$; where k is the so called encoder's constraint length, obviously $k = (\Gamma + 1)\ell$.

To keep the hardware complexity to a minimum, it is desirable to find a subgenerator of minimal size which is called the minimal subgenerator, designated as G_δ . The inverse of G_δ is called the

minimal inverse and designated as G_{δ}^{-1} . Once G_{δ}^{-1} is obtained, the decoding process becomes fairly simple. The received sequence may be shifted into $n(\delta-1)$ stage shift registers one block, i.e. n bits, at a time. After each block is entered into the input shift register, the estimated encoder inputs, $\tilde{X}$, are formed by $(\delta l + v) = (\delta + \Gamma)l \bmod 2$ adders and a connection matrix equal to G_{δ}^{-1} , so that $\tilde{X} = rG_{\delta}^{-1}$, Fig. 4.7. Obviously, $\delta + \Gamma$ blocks of input data are estimated by each decoding operation and only δ operations are participated in by each received block. This leaves at least 1 reliable operation for error correction if the burst length is not over Γ blocks in a span of $\Gamma + \delta$ blocks of the received sequence. However, the most difficult problem is the actual identification of the reliable operation or operations should an error event occur. Most of the discussions in Section 4.3.3 are directed to this problem.

4.2 Physical Realization of Convolutional Encoders

The parameters μ and v mentioned in the last section can serve as measures of the complexity of a convolutional encoder. It is known that any $v = k-l$ rows in the generator matrix can at most span a μ dimensional space, where $\mu \leq v$ with equality holds when the code is nonsystematic. Physically v represents the total number of memory elements in the obvious realization of the encoder. Since it is most likely that a systematic coder can be further simplified, the number of memory elements demanded in its minimal realization is designated a μ . Another interpretation of v and μ is that $\mu = \log_2$ (number of internal state) and $v = \log_2$ (number of output states). An output state is the sequence of outputs obtained

at time t or later, given that the inputs at time t or later are all zeros. The number of output states is less than or equal to the number of internal states.

It is hoped that the following example may clear some of the doubts. Consider a (3,2) systematic code generated by the following subgenerators.

$$g(1,1) = [g_0(1,1), g_1(1,1), g_2(1,1)] = (1 \ 1 \ 1)$$

$$g(2,1) = [g_0(2,1), g_1(2,1), g_2(2,1)] = (1 \ 0 \ 1)$$

A matrix formulation is achieved from the subgenerators through use of some auxiliary functions²¹ in the following manner:

$$P_0 = \begin{bmatrix} g_0(1,1) \\ g_0(2,1) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad P_1 = \begin{bmatrix} g_1(1,1) \\ g_1(2,1) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad P_2 = \begin{bmatrix} g_2(1,1) \\ g_2(2,1) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} g(1) \\ g(2) \end{bmatrix} = [I_2][P_0][0][P_1][0][P_2] = \begin{bmatrix} (1 & 0 & 1) \\ (0 & 1 & 1) \end{bmatrix} \begin{bmatrix} (0 & 0 & 1) \\ (0 & 0 & 0) \end{bmatrix} \begin{bmatrix} (0 & 0 & 1) \\ (0 & 0 & 1) \end{bmatrix}$$

$$= [G_0, G_1, G_2] = \begin{bmatrix} (g_{01}^1 & g_{01}^2 & g_{01}^3) \\ (g_{02}^1 & g_{02}^2 & g_{02}^3) \end{bmatrix} \begin{bmatrix} (g_{11}^1 & g_{11}^2 & g_{11}^3) \\ (g_{12}^1 & g_{12}^2 & g_{12}^3) \end{bmatrix} \begin{bmatrix} (g_{21}^1 & g_{21}^2 & g_{21}^3) \\ (g_{22}^1 & g_{22}^2 & g_{22}^3) \end{bmatrix}$$

and the generator matrix may be written as:

$$\underline{G}_{\infty} = \begin{bmatrix} G_0 & G_1 & G_2 & 0 & 0 \\ 0 & G_0 & G_1 & G_2 & 0 \\ 0 & 0 & G_0 & G_1 & G_2 \\ \vdots & & & & \\ \infty & & & & \end{bmatrix} \quad \text{where} \quad \begin{aligned} G_0 &= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \\ G_1 &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\ G_{\Gamma} = G_2 &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

The parity check matrix may be written using auxilliary functions as follows:

$$P_0^T = [1 \ 1] \quad P_1^T = [1 \ 0] \quad P_2^T = [1 \ 1]$$

and

$$\begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix} = \begin{bmatrix} h_0^1 & h_0^2 & h_0^3 \\ h_1^1 & h_1^2 & h_1^3 \\ h_2^1 & h_2^2 & h_2^3 \end{bmatrix} = \begin{bmatrix} P_0^T & I_{n-l} \\ P_1^T & I_{n-l} \\ P_2^T & I_{n-l} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

then,

$$\underline{H}_{\infty} = \begin{bmatrix} H_0 & 0 & 0 & 0 \\ H_1 & H_0 & 0 & 0 \\ H_2 & H_1 & H_0 & 0 \\ \vdots & & & \\ \infty & & & \end{bmatrix} \quad \text{where} \quad \begin{aligned} H_0 &= [1 \ 1 \ 1] \\ H_1 &= [1 \ 0 \ 0] \\ H_{\Gamma} = H_2 &= [1 \ 1 \ 0] \end{aligned}$$

In order to see the physical significance of G and H , it is more convenient to express them in terms of the Huffman delay operator, ⁴⁰ D , which signifies a time delay of one unit. Thus, if

x_1^1 signifies the first input of block 1, then $D^l x_1^1$ signifies the first input bit of block $1 - l$, i.e. the l th block previous to block 1. The formula for the transformation is

$$G(D) = \begin{bmatrix} G_1^1(D), & G_1^2(D), & \dots, & G_1^n(D) \\ G_2^1(D), & G_2^2(D), & \dots, & G_2^n(D) \\ \vdots & & & \\ G_{n-1}^1(D), & G_{n-1}^2(D), & \dots, & G_{n-1}^n(D) \end{bmatrix},$$

$$X(D) = [X^1(D), X^2(D), \dots, X^{n-1}(D)]$$

where

$$G_L^N(D) = g_{0L}^N + D^1 g_{1L}^N + D^2 g_{2L}^N + \dots + D^r g_{rL}^N \quad \text{for } N = 1, 2, \dots, n$$

$$X^L(D) = x_0^L + D^1 x_1^L + D^2 x_2^L + \dots + D^r x_r^L \quad L = 1, 2, \dots, n-1,$$

and

$$H(D) = [H^1(D), H^2(D), \dots, H^n(D)]$$

$$\text{where } H^N(D) = h_0^N + D^1 h_1^N + \dots + D^r h_r^N \quad \text{for } N = 1, 2, \dots, n.$$

Using the D operator the code can be written as

$$y(D) = x(D)G(D) \quad \text{and} \quad G(D)H(D) = 0$$

By employing the D operator, the example code can be transformed as follows:

$$G_0 = \begin{bmatrix} g_{01}^1 & g_{01}^2 & g_{01}^3 \\ g_{02}^1 & g_{02}^2 & g_{02}^3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \quad G_1 = \begin{bmatrix} g_{11}^1 & g_{11}^2 & g_{11}^3 \\ g_{12}^1 & g_{12}^2 & g_{12}^3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$G_2 = \begin{bmatrix} g_{21}^1 & g_{21}^2 & g_{21}^3 \\ g_{22}^1 & g_{22}^2 & g_{22}^3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} G_1^1(D) &= g_{01}^1 + D g_{11}^1 + D^2 g_{21}^1 & G_1^2(D) &= g_{01}^2 + D g_{11}^2 + D^2 g_{21}^2 \\ &= 1 + 0 + 0 = 1 & &= 0 + 0 + 0 = 0 \end{aligned}$$

$$\begin{aligned} G_1^3(D) &= g_{01}^3 + D g_{11}^3 + D^2 g_{21}^3 \\ &= 1 + D + D^2 \end{aligned}$$

$$\begin{aligned} G_2^1(D) &= g_{02}^1 + D g_{12}^1 + D^2 g_{22}^1 & G_2^2(D) &= g_{02}^2 + D g_{12}^2 + D^2 g_{22}^2 \\ &= 0 + 0 + 0 = 0 & &= 1 + 0 + 0 = 1 \end{aligned}$$

$$\begin{aligned} G_2^3(D) &= g_{02}^3 + D g_{12}^3 + D^2 g_{22}^3 \\ &= 1 + 0 + D^2 \end{aligned}$$

and

$$H_0 = [h_0^1, h_0^2, h_0^3] = [1 \ 1 \ 1] \quad H_1 = [h_1^1, h_1^2, h_1^3] = [1 \ 0 \ 0]$$

$$\begin{aligned} H^1(D) &= h_0^1 + D h_1^1 + D^2 h_2^1 & H^2(D) &= h_0^2 + D h_1^2 + D^2 h_2^2 \\ &= 1 + D + D^2 & &= 1 + D^2 \end{aligned}$$

$$H_2 = [h_2^1, h_2^2, h_2^3] = [1 \ 1 \ 0]$$

$$\begin{aligned} H^3(D) &= h_0^3 + D h_1^3 + D^2 h_2^3 \\ &= 1 \end{aligned}$$

therefore

$$G(D) = \begin{bmatrix} G_1^1(D) & G_1^2(D) & G_1^3(D) \\ G_2^1(D) & G_2^2(D) & G_2^3(D) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 + D + D^2 \\ 0 & 1 & 1 + D^2 \end{bmatrix}$$

$$H(D) = [H^1(D) \ H^2(D) \ H^3(D)] = [1 + D + D^2 \ 1 + D \ 1] .$$

By using the equation $y(D) = x(D)G(D)$, it can be obtained that

$$\begin{aligned} [y_1 \ y_2 \ y_3] &= [x_1 \ x_2] \begin{bmatrix} 1 & 0 & 1 + D + D^2 \\ 0 & 1 & 1 + D^2 \end{bmatrix} \\ &= [x_1 \ x_2 \ (1 + D + D^2)x_1 + (1 + D^2)x_2] \\ &= [x_1 \ x_2 \ (x_1 + x_2) + D x_1 + D^2(x_1 + x_2)] . \end{aligned}$$

Therefore the obvious realization for the encoder is shown in Fig. 4.1.

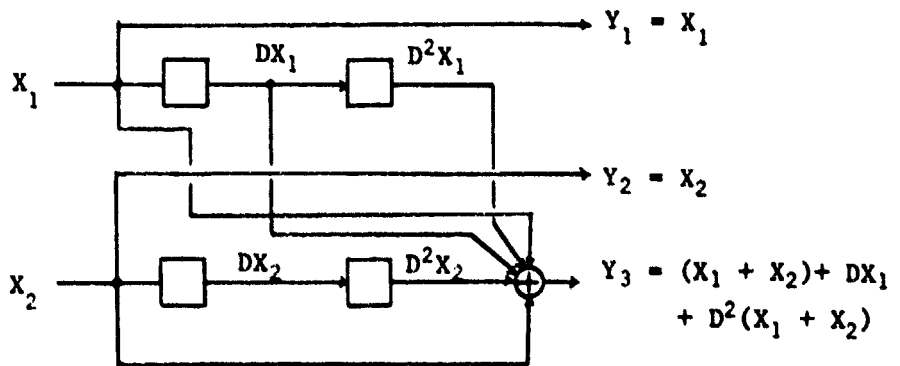


Fig. 4.1 Obvious encoder realization.

From Fig. 4.1 it can be seen that

v_1 = the length of the first shift register, or
the first constraint length

$$= \max \{ \deg[G_1^j(D)] \} = \max \{ \deg[G_1^1(D), G_1^2(D), G_1^3(D)] \}$$

$$1 \leq j \leq 3$$

$$= \max \{ \deg[1, 0, 1 + D + D^2] \} = 2$$

$$v_2 = \max \{ \deg[0, 1, 1 + D^2] \} = 2$$

The overall constraint length v is given by $v = \sum_{i=1}^g v_i = v_1 + v_2 = 4$; therefore, the encoder has $2^v = 2^4 = 16$ internal states. A detailed study of this circuit indicates that there are only $4 = 2^\mu$ distinct output states; therefore, this encoder can be simplified using only $\mu = 2$ memory elements. An encoder using the minimum possible number (μ) of memory elements is called a minimal realization. The minimal realization of the example code is

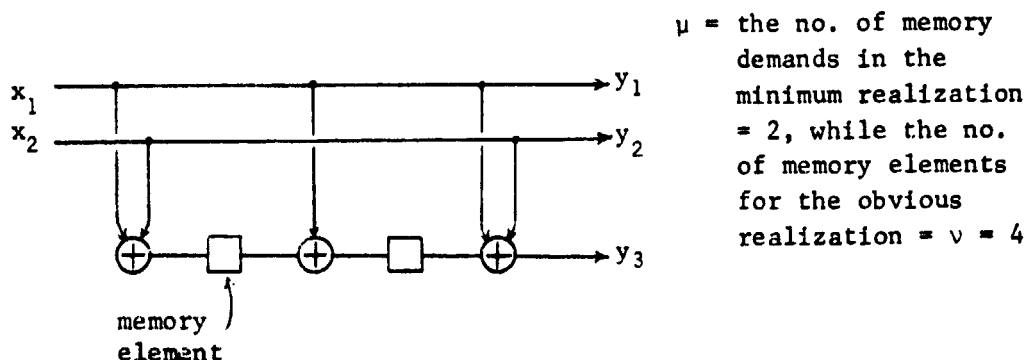


Fig. 4.2 Minimal encoder realization.

As pointed out earlier, $\mu < v$ is a typical phenomenon for the systematic codes and there are only μ linearly independent rows among any v rows in the generator matrix G . This implies that the subgenerator G_s is generally singular or does not have full rank for systematic codes. Therefore, the proposed algebraic decoding does not apply directly to this class of codes unless their generator matrices are simplified according to their minimal realization. The applicability of the proposed decoding method to systematic codes needs further investigation. Since nonsystematic codes are generally superior to systematic codes, this problem will not be pursued any further in this work.

4.3 Algebraic Decoding of Nonsystematic Codes

4.3.1 Derivation of the Minimal Subgenerator and Its Inverse

Since the first block of Y depends on the present block and the previous Γ blocks of X , then $v = \Gamma l$ nonzero rows must be prefixed to G prior to truncating it into the minimal subgenerator G_0 . This, in turn, requires the input sequence X to be prefixed by $v = \Gamma l$ all zero bits.

Consider the Paaske's $(3,2)$, $v = 4$ code whose $\Gamma + 1$ submatrices are

$$G_0 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad G_1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad G_\Gamma = G_2 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

This code has $\mu = 4 = v$ and the rank of G_0 is $2 = l$; apparently it does not violate the conditions of having a minimal

subgenerator G_δ such that G_δ^{-1} exists. The order of G_δ must satisfy $N_\delta = \delta n \geq \left(\frac{v}{n-l}\right)n = \left(\frac{4}{3-2}\right)(3) = 12$.

Generally G_δ may not be a square matrix; G_δ is a square matrix only when v is an integral multiple of $(n-l)$ and δ satisfies its required relation as an equality. More precisely, δn is the number of columns and $(\delta l + v)$ is the number of rows in G_δ ; they are equal only if G_δ is a square matrix.

For the Paaske's (3,2), $v = 4$ code

$$G_\delta = \begin{array}{c} \left. \begin{array}{l} \Gamma \text{ blocks} \\ \text{or } v \text{ bits} \\ \text{prefix} \end{array} \right\} \begin{array}{c} \xrightarrow{\delta n \text{ columns}} \\ \begin{array}{cccccccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \left(\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 1 \end{array} \right) & \left(\begin{array}{cc|c} 1 & 0 & 0 \\ 1 & 0 & 1 \end{array} \right) & \left(\begin{array}{cc|c} 1 & 1 & 0 \\ 0 & 1 & 1 \end{array} \right) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array} \\ \xrightarrow{\delta l + v \text{ rows}} \end{array} \end{array}$$

Then G_δ^{-1} is

$$G_{\delta}^{-1} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

and decoding we would perform $\tilde{x} = r_{\delta} G^{-1}$.

4.3.2 An Example--the (2,1), k = 7 Code

To establish the requirements for the (2,1) k = 7 code we have:

$$\begin{aligned} G_0 &= [1 \ 1] & G_1 &= [1 \ 0] & G_2 &= [1 \ 1] & G_3 &= [1 \ 1] \\ &= [g_{01}^1 g_{01}^2] & &= [g_{11}^1 g_{11}^2] & &= [g_{21}^1 g_{21}^2] & &= [g_{31}^1 g_{31}^2] \\ G_4 &= [0 \ 0] & G_5 &= [0 \ 1] & G_6 &= [1 \ 1] \\ &= [g_{41}^1 g_{41}^2] & &= [g_{51}^1 g_{51}^2] & &= [g_{61}^1 g_{61}^2] \end{aligned}$$

$$\begin{aligned} G_1^1(D) &= g_{01}^1 + D^1 g_{11}^1 + D^2 g_{21}^1 + D^3 g_{31}^1 + D^4 g_{41}^1 + D^5 g_{51}^1 + D^6 g_{61}^1 \\ &= 1 + D^1 + D^2 + D + 0 + 0 + D^6 \end{aligned}$$

$$\begin{aligned}
 G_1^2(D) &= g_{01}^2 + D^1 g_{11}^2 + D^2 g_{21}^2 + D^3 g_{31}^2 + D^4 g_{41}^2 + D^5 g_{51}^2 + D^6 g_{61}^2 \\
 &= 1 + 0 + D^2 + D^3 + 0 + D^5 + D^6
 \end{aligned}$$

and

$$G(D) = [G_1^1(D), G_1^2(D)] = [1 + D + D^2 + D^3 + D^6, 1 + D^2 + D^3 + D^5 + D^6]$$

The minimum realization (equivalent to obvious realization) is:

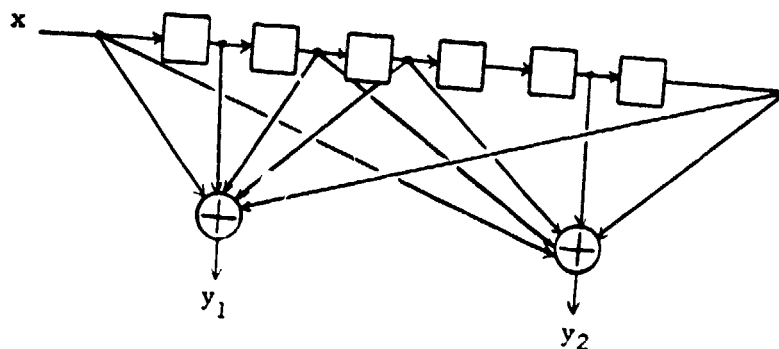


Fig. 4.3 Encoder for the $(2,1)$, $k = 7$ code.

$$v = 6 = \mu$$

$$\delta \geq \frac{v}{n-l} = \frac{6}{1} = 6, \quad \text{and} \quad \text{Rank}\{G_0\} = l = 1$$

Therefore G_δ has $N_\delta = \delta n = 6 \times 2 = 12$ columns

and $\delta l + v = 12$ rows.

Furthermore, G_δ must be prefixed with $v = r l = 6$ nonzero rows.

The minimal subgenerator matrix is:

$$G_{\delta} = \begin{bmatrix} 1 & 1 & & & & & & & & & & & \\ 0 & 1 & 1 & 1 & & & & & & & & & \\ 0 & 0 & 0 & 1 & 1 & 1 & & & & & & & \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & & & & & \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & & & \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & \\ & & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & \\ & & & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & \\ & & & & & 1 & 1 & 1 & 0 & 1 & 1 & \\ & & & & & & 1 & 1 & 1 & 0 & & \\ & & & & & & & 1 & 1 & 1 & 0 & \\ & & & & & & & & 1 & 1 & & \\ & & & & & & & & & 1 & 1 & \end{bmatrix}$$

In order for G_{δ}^{-1} --the inverse of the minimum subgenerator matrix--to exist, three conditions must be satisfied: (1) G_{δ} must have at least δn columns, (2) μ must equal to v , (3) $\text{rank}\{G_0\}$ must be ℓ .

It has been shown that the (2,1), $k = 7$ code under consideration satisfies all these conditions.

With a computer program, using Gaussian elimination technique and mod-2 arithmetic, it was found that the desired inverse is:

Then the encoded sequence would be

$$y = (\underbrace{0000000}_{\substack{\Gamma \text{ bits} \\ \text{prefix}}} 011011011011)[G_\delta]$$

$$= \begin{array}{cccccccccccc} \underline{00} & \underline{11} & \underline{01} & 01 & 11 & 10 & 00 & 01 & 01 & 00 & 01 & \underline{01} \\ y^0 & y^1 & y^2 & . & . & . & . & . & . & . & . & y^{11} \end{array}$$

The decoding process is simply shifting the received sequence into the decoder's input register, one block at a time, and postmultiplying the contents of the register by G_δ^{-1} . Each operation estimates one block of input data and at the same time, several previous blocks. In other words, one block of input data is estimated by several operations. As will be seen in the following developments, it is these repetitions which allow for error detection and correction.

The following table describes the consequence of the decoder operations.

It is easy to see from Table 4.1 that each message block x^j has been estimated $\delta + \Gamma$ times. If the channel is clean, all the $\delta + \Gamma$ estimates should be in perfect agreement. This fact provides us a clue for error detection. The simplest approach is to compare each decoded bit with the same decoded bit obtained from the previous decoder operation. If the two bits disagree, an error is assumed. This is the error detection method, which may be regarded as an alternative way of parity checking.

According to a theorem developed by R. W. Boyd,²⁰ a parity check on the received sequence is obtained from mod-2 addition of the

TABLE 4.1
A SEQUENCE OF DECODER OPERATIONS

Decoder Operation	Contents of Input Register (Y)						Output of Mod 2 Adder (X)											
	r^6	r^7	r^8	r^9	r^{10}	r^{11}	x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^{10}	x^{11}
0	00	00	00	00	00	00	0	0	0	0	0	0	0	0	0	0	0	0
1	00	00	00	00	00	11	0	0	0	0	0	0	0	0	0	0	0	1
2	00	00	00	00	11	01	0	0	0	0	0	0	0	0	0	0	1	1
3	00	00	00	11	01	01	0	0	0	0	0	0	0	0	1	1	0	0
4	00	00	11	01	01	11	0	0	0	0	0	0	0	0	1	1	0	1
5	00	11	01	01	11	10	0	0	0	0	0	0	0	1	1	0	1	1
6	11	01	01	11	10	00	0	0	0	0	0	0	1	1	0	1	1	0
7	01	01	11	10	00	01	0	0	0	0	0	1	1	0	1	1	0	1
8	01	11	10	00	01	01	0	0	0	0	1	1	0	1	1	0	1	1
9	11	10	00	01	01	00	0	0	0	1	1	0	1	1	0	1	1	0
10	10	00	01	01	00	01	0	0	1	1	0	1	1	0	1	1	0	1
11	00	01	01	00	01	01	0	1	1	0	1	1	0	1	1	0	1	1

output associated with column ξ for one decoder operation and the output associated with column $\xi - 1$ for the next operation, providing column ξ is one of the last $(\delta - 1) \ell$ columns of G_{δ}^{-1} and is nonzero in at least one of its first n positions.

For the example code ξ shall be chosen from the last $(\delta - 1) \ell = 5$ columns of G_{δ}^{-1} such that its first $n = 2$ positions are not all zeros. Column 11 of G_{δ}^{-1} fits the theorem and thus the existence of errors can be detected by mod-2 addition of the estimate associated with col. 10, i.e. $\xi - 1$, with the estimate associated with col. 11, i.e. ξ , obtained from the previous operation. The result of this comparison is appended to the syndrome sequence. An all zero syndrome indicates that no channel errors have occurred.

Since one block of the input data has been estimated by several decoding operations, error corrections can be achieved by replacing the estimate obtained from the erroneous block with any estimate for the same bit which did not require the use of that block.

It has been shown that a message block j , say x^j , will be estimated $\delta + \Gamma$ times; however, an error in x^j can only propagate $\delta - 1$ times. This fact indicates that operation 1 may be used to correct an error in block j , say x^j , as long as $j + \delta \leq i < j + \delta + \Gamma$.

The present approach is to use the syndrome sequence to signal the beginning of an error rather than to identify an error pattern. When an error starts, beginning with that block, the next $\delta - 1$ blocks of estimate must not be used for error correction as error effects are still contained in these estimates. The Γ estimates following the $\delta - 1$ estimates are supposed to be error free

providing that successive erroneous blocks are not closer than $\delta + \Gamma - 1$ blocks, i.e. a guard space of $\delta + \Gamma - 1$ blocks are required. This is to safeguard the estimates for block j obtained from operations $j + \delta$ to $j + \delta + \Gamma - 1$ are not corrupted by other erroneous received blocks having direct effects on this interval.

Since an error in the r^j , may not cause a "1" immediately for the syndrome bit associated with that operation, i.e. s^j may not be "1", a method must be devised to keep track of the beginning of a syndrome even if the first few syndrome bits are zeros.

A list of all the possible error patterns in one received block and their associated syndrome sequences is given in the following table.

Event	Error Pattern in Block j		Syndrome							
			s^j	s^{j+1}	s^{j+2}	s^{j+3}	s^{j+4}	s^{j+5}	s^{j+6}	s^{j+7}
A	0 1		1	1	1	1	0	0	1	0 ...
B	1 0		1	0	1	1	0	1	1	0 ...
C	1 1		0	1	0	0	0	1	0	0 ...

The syndromes are listed to the point where the effects of the errors have ended.

From the above table, it is clear that when the first "1" appears in a syndrome sequence, errors may have already propagated several blocks. For instance, for event C, the earliest possible error indicated by $s^0 = 1$ has $j = -1$ which means that an error in block r^{-1} shows its nonzero syndrome bit after receiving block r^0 . This implies that this error can be corrected by operation 1 where

$$-1 + \delta \leq i < -1 + \delta + \Gamma, \text{ or } 5 \leq i < 11,$$

while both events A and B have $s^0 = 1$ with $j = 0$, meaning they can be corrected by operation i , where $6 \leq i < 12$.

In order to distinguish the various error patterns that require different correction timing since the first appearance of the nonzero syndrome, it seems that the entire syndrome sequences have to be used. A second look shows that this is not so. As can be seen easily from the syndrome table, if the first two syndrome bits are 11, event A is uniquely identified. If the first two bits are 10, correct identification of error events depends on one more syndrome bit; if the third bit is a "1", event B is the error; if a "0", event C is the one. A systematic way of doing this is to draw a "syndrome tree" as shown in Fig. 4.4. The tree stops at the point where each branch corresponds to only one possible error event. (The notation C^{-1} signifies error pattern C in block r^{-1}). Events with the same

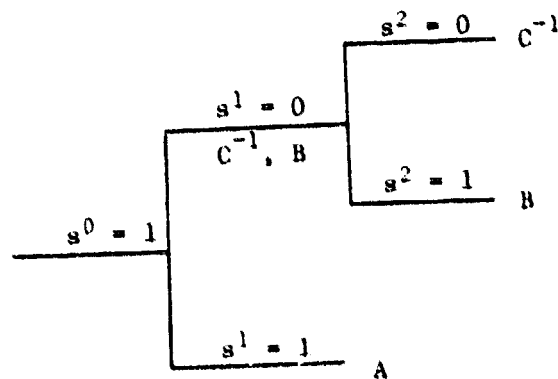


Fig. 4.4 Syndrome tree.

"-1"s are grouped together and a syndrome sequence shall be recorded to the point that each group can be uniquely identified. For the example code,

	s^0	s^1	s^2	s^3	s^4	s^5	s^6	
If	$s = 1$	0	0	-	-	-		correct and reset s - group 1
	$s = 1$	-	-	-	-	-		correct and reset s - group 2

where "-" indicates "don't care". One way to implement this algorithm is to use the following circuitry to match the syndrome sequences obtained above for each group of error events so that a "1" will be produced to signal the correct timing for error correction.

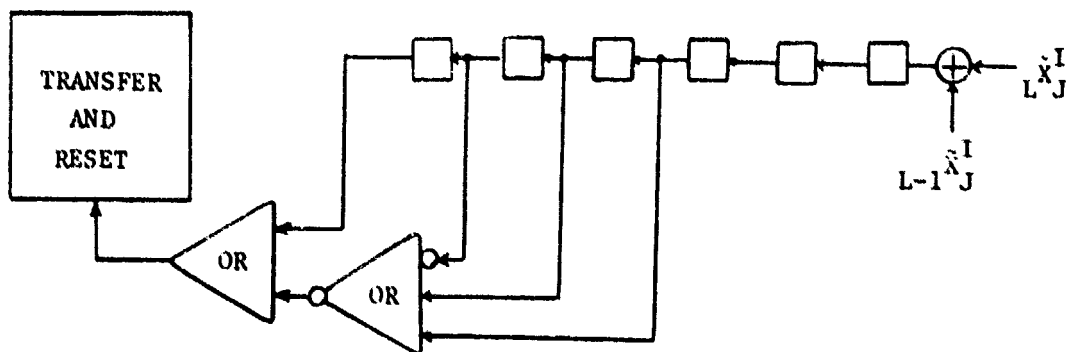


Fig. 4.5 Syndrome search logic.

The decoder consists of a connection matrix which corresponds to G_s^{-1} , $n(\delta - 1)$ -stages input registers, $(v + \delta\ell) \bmod 2$ adders, $\ell(v + \delta\ell - 1)$ - stage output registers and a syndrome search logic

as shown in Fig. 4.5.

Decoding operations are as follows: the received sequence is shifted into the input registers one block at a time. After each block is entered into the input register, the estimated encoder inputs $\tilde{x}_1$ are formed by the $(\delta t + v) \bmod 2$ adders and the connection matrix. This action is equivalent to performing $\tilde{x} = \delta G_\delta^{-1}$. If there are no erroneous received bits, the estimates will be exact, i.e. $\tilde{x} = x$, and the syndrome bit that corresponds to that block will be zero. However, if the received block is in error, beginning from that operation and lasting through the next $(\delta - 1)$ operations, the estimates for that block will be unreliable. Under the condition that the following $\delta + \Gamma - 1$ input blocks are all clean, estimates for that block obtained from the next Γ operations are correct. The tracking of the errors is taken care of by the syndrome search logic. If channel errors cause incorrect estimates to enter the output shift register, the condition is indicated by a nonzero syndrome and the erroneous estimates are replaced by other estimates via the transfer gates as soon as correct estimates are obtainable, i.e. when the syndrome search logic produces a "1" output, and the syndrome register is reset to zero to restart another search.

Fig. 4.5.

4.3.4 Modified Algebraic Decoder

The above proposed decoding method works well when bursts are separated by enough error free blocks; its estimates are unreliable if the channel is not so benevolent. This motivates consideration of a hybrid decoding system.

Since the checking of the guard space is the main issue, it is necessary to know when an error starts. One way of achieving this purpose is to re-encode the estimated sequence and compare it with the received sequence, block by block. Whenever the two sequences do not agree, that block is considered to contain an error. Since δ error free blocks are required, as soon as an error is detected, the next δ blocks are examined. If they all agree, the decoder assumes an error free guard space was received, which would imply that the error was properly corrected. If the next δ blocks do not agree, the decoder estimates will be regarded as unreliable and a second algorithm will have to be used. Since the algebraic decoder is a good burst corrector and requires an error free guard space of finite length, the second algorithm would be most likely a random error corrector and hence would not require error-free guard space; in particular, the maximum likelihood probabilistic decoder, namely, the Viterbi decoder is proposed as the second phase of the hybrid system.

Additional hardware includes an extra encoder to re-encode the estimated sequence, $n \bmod 2$ adders to compare the re-encoded sequence with the received sequence and a flag logic to check the violation of guard space requirement. Whenever the outputs of the second encoder differ from the received sequence for a certain block, an "1" enters the flag logic; after it propagates δ times, a flag will be set. If there are no differences between the re-encoded sequence and the received sequence in the guard space, indicated by an all zero flag register, the flag is reset to zero and the estimate of that block is regarded as correct. A flagged decoder estimate is considered to be

an erasure and the phase II random error corrector will be summoned to take over the duty.

Since a guard space of $\delta = 6$ blocks is required for the example code, the "flag logic" would be as shown in Fig. 4.6.

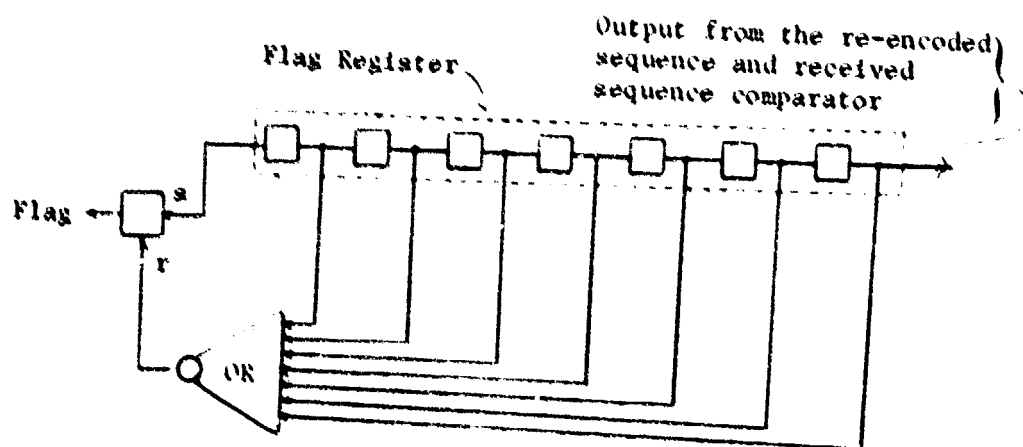


Fig 4.6 Flag logic.

The idea of comparing the re-encoded sequence with the received sequence to check the existence of the required clean-guard space is not very reliable. Although clean intervals always agree with the received sequence when re-encoded, the converse is not necessarily true; an erroneous block may also agree with its corresponding received block when re-encoded if the error bits are not corrected. As a matter of fact, a whole burst might be mistaken as a clean interval if no correction had ever been made. Therefore, the modified

algebraic decoder works well only for the correction of short bursts where the burst corrections can be made definite.

More will be said about the hybrid system and several alternatives will be presented in the next chapter.

4.3.5 Probabilistic Decoding of Long Bursts

So far discussion has been limited to the correction of short bursts of one block. The definitions of "long bursts" or "short bursts" are not intended to be rigorous, those bursts that can be stored in a decoding table or incorporated into the syndrome search logic so that their decodings can be made definite may be defined as short bursts; otherwise, as long bursts. Therefore short bursts can be regarded as bursts of one to two blocks. As pointed out earlier, the multiple parity check encoding allows one source block to be decoded $(\Gamma + \delta)$ times while only δ received blocks are required for each decoding operation. This means that if one received block is in error, there are Γ chances to recover the source sequence, i.e. if r^j is in error, the i^{th} estimate can be used for correction as long as $j + \delta \leq i < j + \delta + \Gamma$. With each additional block of the burst length, the chances diminish one. Therefore, the entire source sequence of $(\Gamma + \delta)\ell$ bits is still recoverable as long as the burst length does not exceed Γ blocks. Thus, Γ consecutive bursty blocks, beginning at the j^{th} block, could be corrected by the $(j + \delta + \Gamma - 1)^{\text{th}}$ estimate, providing that the following received blocks, i.e. $r^{j+\Gamma}$ to $r^{j+\Gamma+\delta-1}$ blocks, are all error free. The potential burst correcting capability of the algebraic decoder is that a burst of length Γ blocks can be corrected as long as it is

TABLE 4.2
DECODER OPERATIONS IN THE PRESENCE OF A LONG BURST

Source sequence:	C	1	1	0	1	1	0	1	1	0	1	1
Code sequence:	00	11	01	01	11	10	00	01	01	00	01	01
Error sequence:	11	11	11	11	11	11	00	00	00	00	00	00
Received sequence:	11	00	10	10	00	01	00	01	01	00	01	01
Decoded sequence:	0	1	1	0	1	1	0	1	1	0	1	1

Decoder Operation	Content of Input Register (r)						Output of Mod-2 Adder (s)															
0	00	00	00	00	00	$\frac{11}{r^0}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{1}{s^0}$
1	00	00	00	00	$\frac{11}{r^0}$	$\frac{00}{r^1}$	1	1	1	1	0	1	1	1	0	1	1	$\frac{1}{s^0}$	$\frac{1}{s^1}$	$\frac{1}{s^2}$	$\frac{1}{s^3}$	$\frac{1}{s^4}$
2	00	00	00	$\frac{11}{r^0}$	$\frac{00}{r^1}$	$\frac{10}{r^2}$	0	0	0	1	1	0	0	1	1	0	$\frac{0}{s^0}$	$\frac{1}{s^1}$	$\frac{1}{s^2}$	$\frac{1}{s^3}$	$\frac{1}{s^4}$	$\frac{1}{s^5}$
3	00	00	11	00	10	10	1	1	0	0	0	1	0	0	0	0	0	1	1	1	1	$\frac{1}{s^0}$
4	00	11	00	10	10	00	0	1	1	1	1	1	1	1	1	0	0	1	0	0	0	$\frac{1}{s^0}$
5	11	00	10	10	00	01	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0	$\frac{1}{s^0}$
6	00	10	10	00	01	00	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	$\frac{1}{s^0}$
7	10	10	00	01	00	01	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	$\frac{1}{s^0}$
8	10	00	01	00	01	01	1	1	1	0	1	0	1	1	1	1	1	0	0	0	0	$\frac{1}{s^0}$
9	00	01	00	01	01	00	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	$\frac{1}{s^0}$
10	01	00	01	01	00	01	1	0	1	1	0	1	1	0	1	1	0	1	0	1	1	$\frac{1}{s^0}$
11	$\frac{00}{r^0}$	$\frac{01}{r^1}$	$\frac{01}{r^2}$	$\frac{00}{r^3}$	$\frac{01}{r^4}$	$\frac{01}{r^5}$	$\frac{0}{s^0}$	$\frac{1}{s^1}$	$\frac{1}{s^2}$	$\frac{0}{s^3}$	$\frac{1}{s^4}$	$\frac{1}{s^5}$	$\frac{0}{s^6}$	$\frac{1}{s^7}$	$\frac{1}{s^8}$	$\frac{0}{s^9}$	$\frac{1}{s^{10}}$	$\frac{1}{s^{11}}$	$\frac{0}{s^{12}}$	$\frac{1}{s^{13}}$	$\frac{1}{s^{14}}$	$\frac{1}{s^{15}}$

followed immediately by a clean guard space of δ blocks. The ratio of guard space required to the maximum burst length correctable is simply δ/Γ . For the example (2,1), $k = 7$ code considered, this ratio is 1; therefore, it obviously outperforms the Wyner-Ash's codes and is justified as an optimal B2 code.

Correction of long bursts cannot be made definite because of the large number of syndrome patterns involved. However, the long bursts may still be decoded with high probability of being correct by extracting the general characteristics of the ensemble of syndrome patterns. One method of decoding long bursts is to examine the syndrome sequence to see if a certain length of consecutive zeros can be obtained. As soon as this pre-established criterion is satisfied, the decoder retracks to the operation prior to the appearance of the first zero and replaces the estimates of that operation for the previous $(\Gamma + \delta)$ decoded blocks. This requires a buffer register to store the estimated $(\Gamma + \delta)$ blocks decoded bits at each step and constantly replaces the contents of the buffer register with the current estimates unless the syndrome bit at that step is zero. This decoding method does not always catch the bursts in time; first, the criterion is difficult to establish; if it is made too tight, an incorrect estimation may be mistaken as the correct one. If it is made too loose, correct timing may be missed and a large number of correctable bursts may be left uncorrected. One favorable feature of this method is that, if the clean guard space is long enough, i.e. longer than $(\delta + z)$ blocks, where z is the length of the pre-established all zero syndrome sequence that has to be checked, not

only can most of the long bursts be corrected but most of the longer bursts, regardless of how long they are, can be partially corrected, i.e. at least the last r blocks of most longer bursts can be corrected.

4.4 Error Bound and Characteristics of the Decoder

Calculation of error bound is extremely difficult due to the large number of error patterns that could be decoded probabilistically. However, in order to provide some indication of the performance of the decoder, a rough estimate is attempted which could be a very loose measure of its capability.

Since the proposed decoder is capable of correcting all short bursts of one to two blocks definitely, with some capability for longer bursts of any length, the overall error probability should be upper bounded by the probability of getting bursts longer than 2 blocks.

Let p be the channel error rate. Then $q = 1 - p$ would be the probability of any one bit being received correctly. A long burst can be viewed as a short burst whose guard space contains at least one erroneous bit. The probability of such events is $P(\Phi) = (1 - q^4)(1 - q^{12})$. An incorrect decoding will output 12 uncertain bits; if assuming random binary values for each bit, only half of them could be decoded incorrectly. Thus, the overall bit error probability is upper bounded by

$$P(E) \leq \frac{12}{2} (1 - q^4)(1 - q^{12})$$

Clearly, this is a very loose bound, since it is assumed that none of the long bursts can be corrected. Of course, this is not true with the present algorithm.

The circuit of the entire decoder was given in Fig. 4.7. It is quite simple. Parts count is around 4 times the number of rows, with the addition of two buffer registers if long bursts are to be corrected--one for the storage of outputs for later retrieval and one for temporarily holding the contents of the transfer gate for later use.

The decoder operates quickly if only bursts of one to two blocks are stored in the table, or incorporated in the syndrome search logic for definite decoding. For long bursts correction, after an initial delay of a few blocks, which is compatible with the requirement of the Viterbi decoder, the rest of the decodings are in real time and no real disadvantages will result.

Since G_{δ}^{-1} is of finite size, error propagation can be greatly suppressed. For the example code, the error effect of any received block cannot propagate to more than $\delta = 6$ consecutive operations from the time it enters the decoder. This is a great advantage of the algebraic decoder over the probabilistic algorithms.

As mentioned in Sec. 2.4, the Wyner-Ash criterion of optimal B2 code is

$$\frac{\text{length of guard space}}{\text{length of longest burst correctable}} = \frac{n + \ell}{n - \ell}$$

For the (2,1), $k = 7$ example code, the ratio is 3. Since this code is capable of correcting bursts of 4 bits when a guard space of 12

The overall algebraic decoder can be shown as follows:

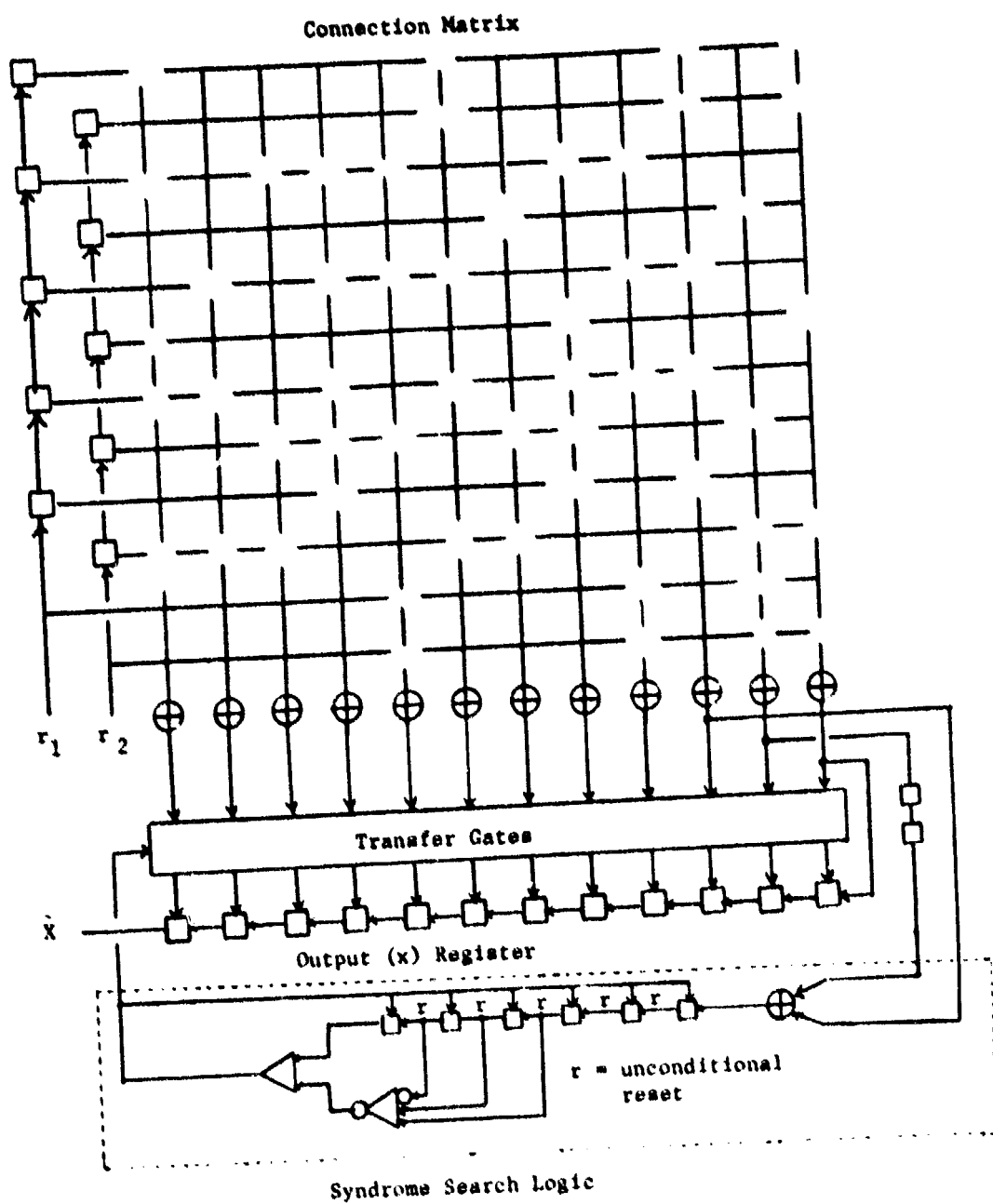


Figure 4.7 Error-Correcting Decoder.

bits is given, it satisfies the above equation exactly. It may be concluded that this code is indeed an optimal B2 burst corrector and the proposed decoder is fully capable of exploiting its bursts-correcting capability. What makes this achievement significant is: this optimal burst corrector is obtained from an optimal d_{free} code, the best random error correcting code. Experiments have found it to be highly likely that the proposed algebraic decoder works this way for any optimal d_{free} codes. This assertion depends on further understanding of the characteristics of the optimal d_{free} codes. It can be seen from Wyner-Ash's formula that for fixed $\frac{n}{l}$, the longer the constraint length of the coder, the longer the bursts can be corrected; however, the guard space required increases proportionally. Furthermore, the larger the r , or equivalently k , the more flexible the algorithm; i.e. there are more operations available for corrections. Therefore, longer constraint codes make the correction of long bursts easier. This is consistent with the requirement of being a good random error corrector.

4.5 Conclusion

The goal of developing an algebraic decoder which is general enough to be applied to nonsystematic codes has been achieved in this chapter.

It has been shown that this algebraic decoder performs extremely well for the example $(2,1)$, $k = 7$ code; it is capable of correcting short bursts of one to two blocks definitely if a table look-up type of scheme is used. It will be verified empirically in Chapter V that even long bursts up to 6 blocks can be corrected with

high probability of success. Corrections of bursts longer than 6 blocks are limited to their last 6 blocks. The only requirement is that 6 blocks of clean guard space must follow immediately after the short bursts with somewhat longer space for the long bursts. If the syndrome table can be made to include all the burst patterns, 6 blocks of guard space would be all that is needed for any bursts, long or short. The relative short guard space required as compared to its burst error correcting capability justifies the $(2,1)$, $k = 7$ code as an optimal B2 corrector in the sense of the Wyner-Ash criterion.

The conclusion, then, is that if methods could be provided to remove random errors prior to the application of the algebraic decoder, the chance of burst error corrections would be greatly improved as the chance of acquiring clean guard space is increased. This will be the topic of the next chapter.

CHAPTER V

HYBRID DECODER

5.1 Philosophy

Although the algebraic decoder has the potential of correcting any bursts of length Γ blocks or less in a $\Gamma + \delta$ block's span of incoming data stream, the number of distinct syndrome patterns that have to be identified forbids such attempts for large Γ . With the innovation of new memory devices such as magnetic-bubble memories, however, such large memory requirements may not be impossible in the future.

Unlike the concatenated codes, the proposed coding scheme employs only one code, an optimal d_{free} convolutional code, with a two phase decoding process. The interfacing of the two phases is quite flexible. Various probabilistic decoding algorithms can be selected as the second phase and different interfacing methods can be employed to fit the particular channel characteristics and the user's needs.

Previously, an attempt has been made to use the algebraic decoder to carry the main load, the Viterbi decoder coming into help only when the desired error-free guard space does not exist. This scheme has the advantage of being fast and causing relatively short decoding delay. However, if long bursts--i.e., bursts which have not been designed into the syndrome search logic to be identified--are encountered, the hybrid system will lose synchronization and errors will propagate. Study of the performance of the Viterbi decoder in Chapter

III reveals that this decoding algorithm is also capable of correcting short bursts, i.e., bursts of two blocks or less. Therefore, it can be concluded that both the algebraic algorithm and the Viterbi algorithm are capable of correcting random errors and short bursts, but the latter do not require error-free guard space. It is the ability of the two algorithms to handle long bursts that needs our attention. In the Viterbi decoder, which is optimal in the sense that it utilizes the entire distance between codewords available from the encoder, error propagation is much more prominent than in the algebraic decoder, which bases its decoding decision on the truncated data stream. The superiority of the Viterbi decoder in handling random errors and short bursts and the complementary nature of the algebraic decoder and the Viterbi decoder when encountering long bursts, motivated the development of an alternative hybrid decoding system that possesses the same error-correcting capability as the Viterbi decoder during random errors and short bursts but can suppress error propagation effects as well as the algebraic decoder.

In order to achieve such a system, methods must be developed to switch the decoding load back and forth between the two decoders according to the channel property at the moment. If the channel is clean or has random errors or short bursts, the decoding duty is allocated to the Viterbi decoder. If the channel encounters a long burst, as soon as it is certain that the burst is over, the load is shifted to the algebraic decoder for the next 72 bits duration. When the hybrid system is operating in the algebraic mode, should another long burst error occur, that mode is terminated immediately and a shift is made back to the Viterbi decoder. In this manner, no

error-correcting duty is allocated to the algebraic decoder; it is used only to identify the syndrome sequence and as a subsidiary decoding device during the resynchronization period of the Viterbi decoder. As pointed out earlier, the algebraic decoder is not capable of correcting any error that is not correctable by the Viterbi decoder but the latter does have a much longer decoding delay than the former. Thus, to let the algebraic decoder take over the duty only after the long burst is over seems a logical choice.

A flow diagram will depict the overall concept and the hybrid decoding procedure much more clearly, Fig. 5.1.

5.2 Algorithms

5.2.1 Syndrome Detecting Logic

For the alternative hybrid decoding system, syndrome detecting instead of syndrome searching becomes the main theme. As indicated in the flow diagram, the hybrid system must be able to detect the beginning and the ending of a burst and to distinguish the long bursts from the short bursts. One possible implementation of the syndrome detecting logic is shown in Fig. 5.2.

The syndrome detecting logic so designed will produce a "1" if, and only if, when a long burst is over, it stays at "1" for the next 36 operations unless encountering another error which will terminate the algebraic mode prematurely. Thus, the hybrid algorithm will be such that if the syndrome detecting logic output is "0", the Viterbi decoder will be employed; otherwise, the algebraic decoder will be employed.

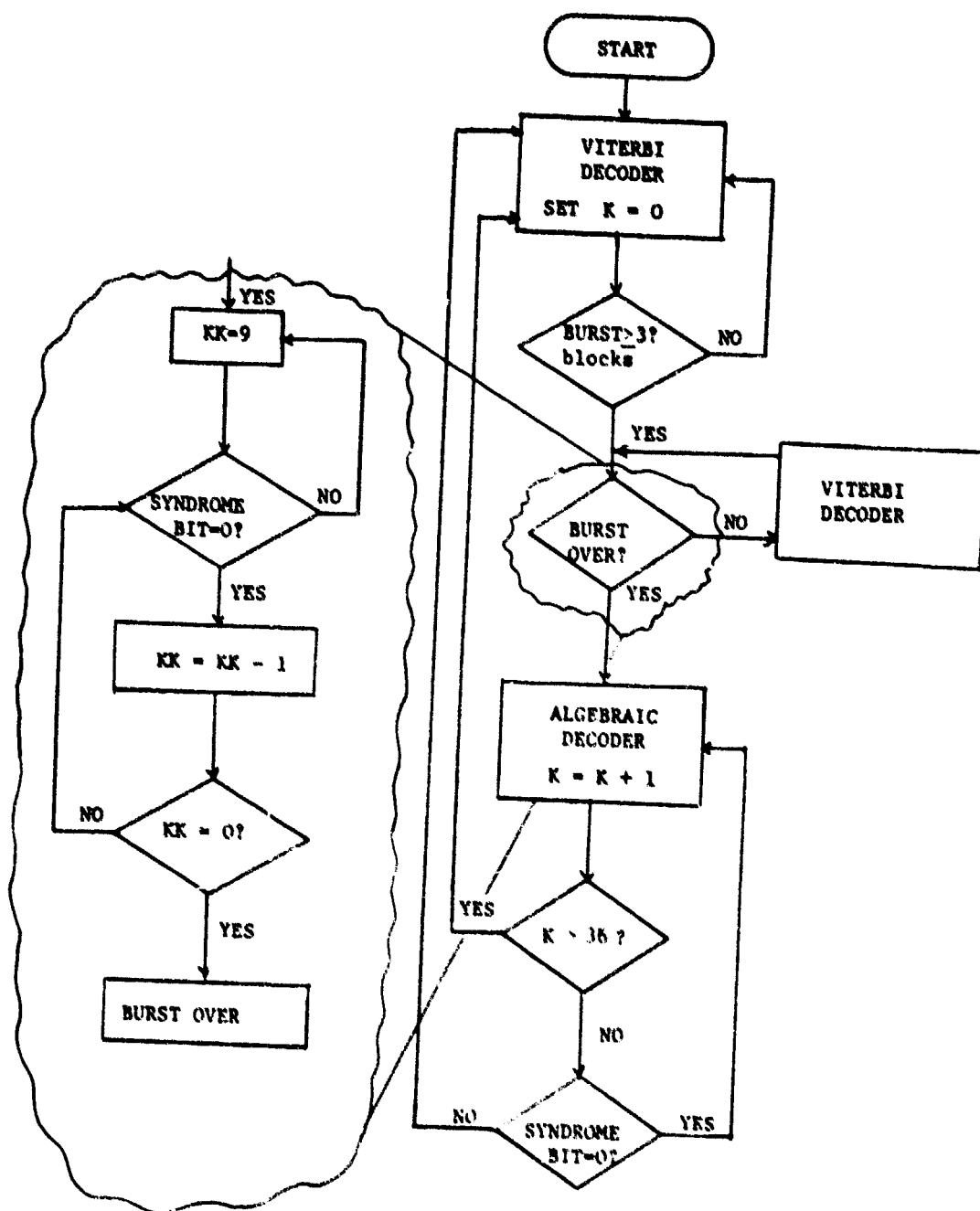


Figure 5.1 Hybrid Decoding Flow Chart

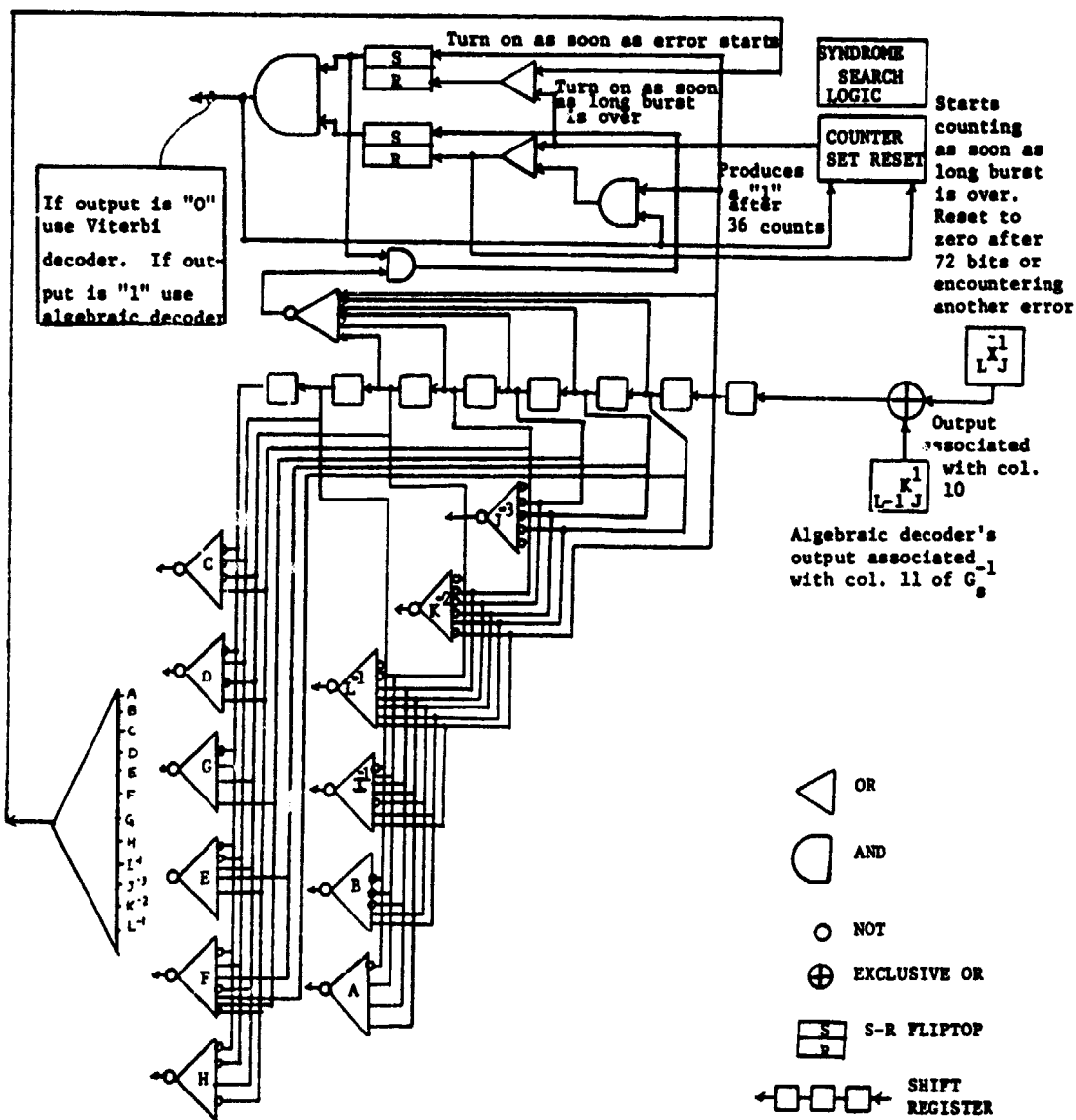


Figure 5.2 Syndrome Detecting Logic

ORIGINAL PAGE IS
OF POOR QUALITY

The development of the syndrome detecting logic is quite complicated, especially the syndrome search portion which involves the listing of all distinct syndrome patterns for short bursts and the forming of a syndrome tree. The purpose of the syndrome tree is to eliminate all the redundant bits of the syndrome sequences under consideration so that a simpler logic system can be made. Further simplification or combination of the "NOR" gates is possible by the "prime implicant table" method.

5.2.2 Syndrome Detecting Logic Development Procedure

The function of the syndrome detecting logic, S.D.L., is to detect long bursts which are not correctable by the Viterbi decoder and to allocate the decoding duty to the algebraic decoder so that error propagation can be confined to the latter.

The heart of the S.D.L. is the syndrome search logic S.S.L., which identifies specific short bursts, should a bursty event occur, therefore allowing recognition of longer bursts not individually recognized.

The first step toward the development of the syndrome search logic is to list all the possible syndrome patterns caused by the short bursts. The best way to achieve this is to use a computer program that simulates the syndrome calculator and tabulate the error vectors and their associated syndrome patterns in the output.

Table 5.1 is a list of the syndrome patterns of the short bursts, i.e. bursts confined to within two blocks, for the example code.

TABLE 5.1

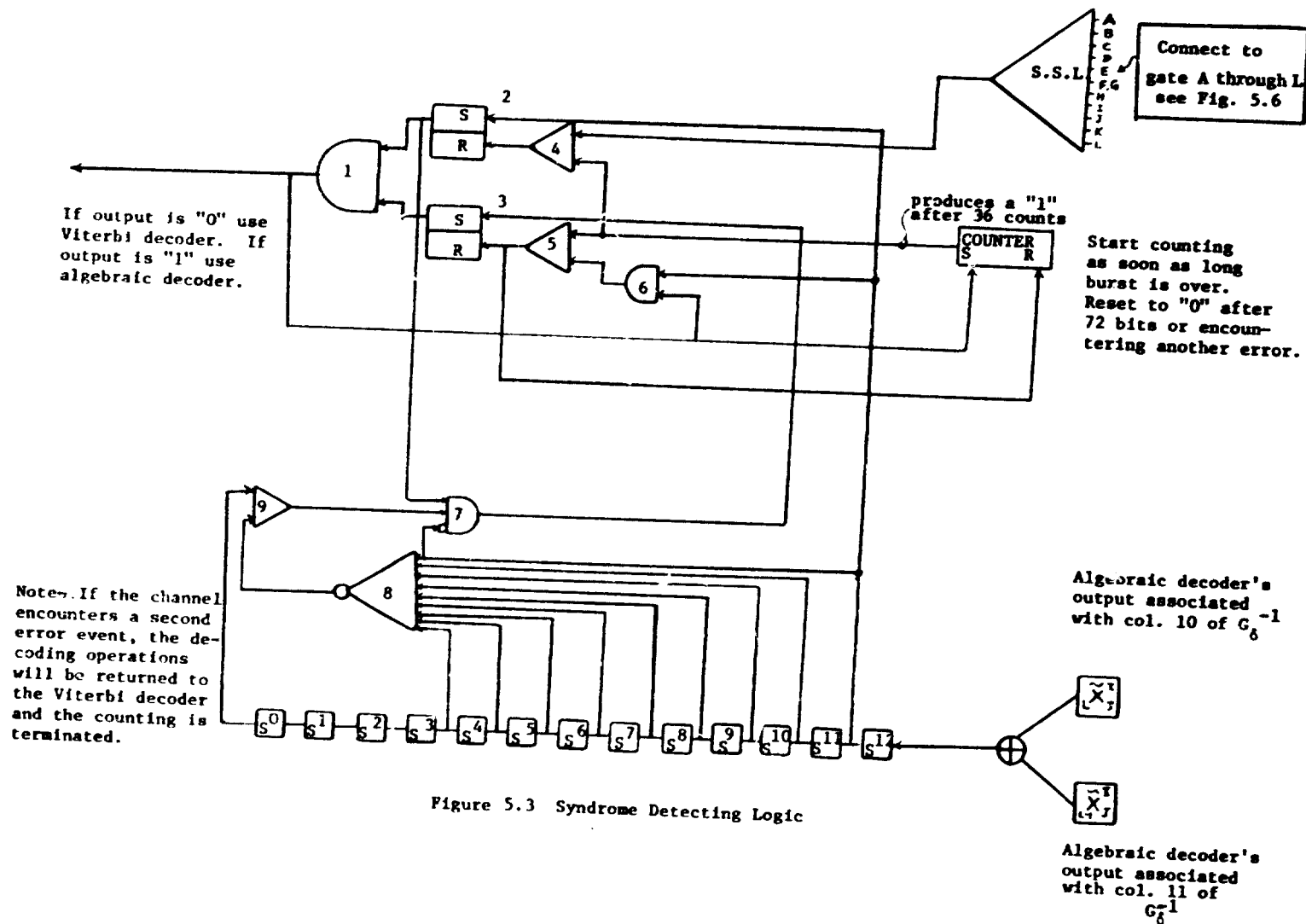
LIST OF THE SYNDROME PATTERNS OF THE SHORT BURSTS

EVENT	ERROR PATTERN				SYNDROME								PATTERN				LENGTH
					s^0	s^1	s^2	s^3	s^4	s^5	s^6	s^7					
A	1	0	0	0	1	0	1	1	0	1	1	0	} May be combined as 1000101-				7
B	0	1	0	0	1	1	1	1	0	0	1	0					7
I^{-1}	1	1	0	0	0	1	0	0	0	1	0	0					5
C	1	0	1	0	1	1	1	0	1	1	0	1					8
D	0	1	1	0	1	0	1	0	1	0	0	1					8
E	1	0	0	1	1	1	0	0	1	1	1	1					8
F	0	1	0	1	1	0	0	0	1	0	1	1					8
G	1	0	1	1	1	0	0	0	1	0	1	0					8
H	0	1	1	1	1	1	0	1	0	0	0	0					4
J^{-3}	1	1	1	0	0	0	0	1	1	1	1	1					5
K^{-2}	1	1	0	1	0	0	1	1	1	1	0	1					6
L^{-1}	1	1	1	1	0	1	1	0	0	1	1	0					6

As may be noticed, the first nonzero digit of the syndrome sequence may not appear immediately after the receipt of an erroneous block, that is, there may be a delay of several input blocks before the first appearance of an "1" in the syndrome sequence. This effect is designated by a superscript; i.e. J^{-3} signifies error pattern J in block r^{-3} . The superscript "0" will be dropped, i.e. $A = A^0$.

In Chapter IV the use of a syndrome tree to eliminate some of the redundant syndrome bits has been mentioned. While it is legitimate to do so if the purpose is to distinguish one short burst from the others, this simplification is not valid if the purpose is to abstract the short bursts from all the possible bursts, long and short. On the contrary, the number of bits to be examined must be extended to a length at least equivalent to the syndrome length of the maximum length bursts under consideration. For the example channel, it is assumed that no burst of length 6 blocks (12 bits) or longer should exist or at least the chances of encountering such bursts are extremely slim. Based upon this assumption, it is evident that the maximum length syndrome patterns are limited to 12 bits. This means syndrome sequence for a 12 bit span must be examined before one can be certain whether it is a short burst or otherwise. The reason is very simple--the syndrome pattern of a short burst can be the prefix of the syndrome pattern of some long bursts and until the entire length of the pattern has been examined no assertions can be made concerning the length of the burst.

Because of the above reasoning, the "syndrome detecting logic" has been revised to that shown in Fig. 5.3. The shift register is



extended to 12 stages with one additional stage which will be explained later. Furthermore, because some syndrome patterns of the long bursts have 8 consecutive zeros, there can be no certainty that a burst is over by observing 7 consecutive zeros. Since the shortest syndrome of the short bursts has 4 bits, the syndrome detecting logic must be implemented in such a way that this syndrome is detected before gate 3 is turned on. This requires that at least 9 consecutive zeros must have entered the shift register before gate 3 can be turned on; otherwise, this shortest burst will always be interpreted as "a long burst is over" for some time, and then behave unpredictably for the next 36 operations. In order to avoid excessive delays after a long burst is over, the "S⁰" stage is added to limit the delay to the least possible. Of course the whole design is based on the assumption that no burst longer than 12 bits should ever occur. If, in fact, the channel encounters a burst of L blocks longer than 6 blocks, the detecting logic will direct the decoding action to the algebraic decoder L bits too early and this will result in L bits data loss; however, this is not a disastrous situation, since bursts longer than 6 blocks will be decoded in error by the Viterbi decoder also, but with the following 72 bit delay as well. The beauty of this logic circuit is that it can catch the ending of most long bursts within 4 operations with only a few exceptions, these being the bursts whose syndromes have excessive delays. However, in no case will the delay be longer than 9 operations, and for an overwhelming majority of the bursts, the ending can be indicated exactly; these are the longest bursts anticipated, i.e. bursts of 6 blocks, though they are not supposed to occur frequently.

Further simplification and combination of the syndrome patterns of short bursts may be possible by the "Karnaugh map" or "prime implicant table" method, but in practice this is highly unlikely. The syndrome patterns of the short bursts consist of only a very small portion of the total possible syndrome patterns and in general they are widely scattered in the "Karnaugh map", i.e. the chance of two syndrome patterns of the short bursts being adjacent is extremely small. For the example code, only the syndromes of event K and event F can be combined with the elimination of one variable.

The rest of the syndrome detecting logic is a direct translation of the verbal specification into logic expressions. This will be self-evident after a detailed explanation of the functioning of the logic circuit.

5.2.1 The Functioning of the Syndrome Detecting Logic

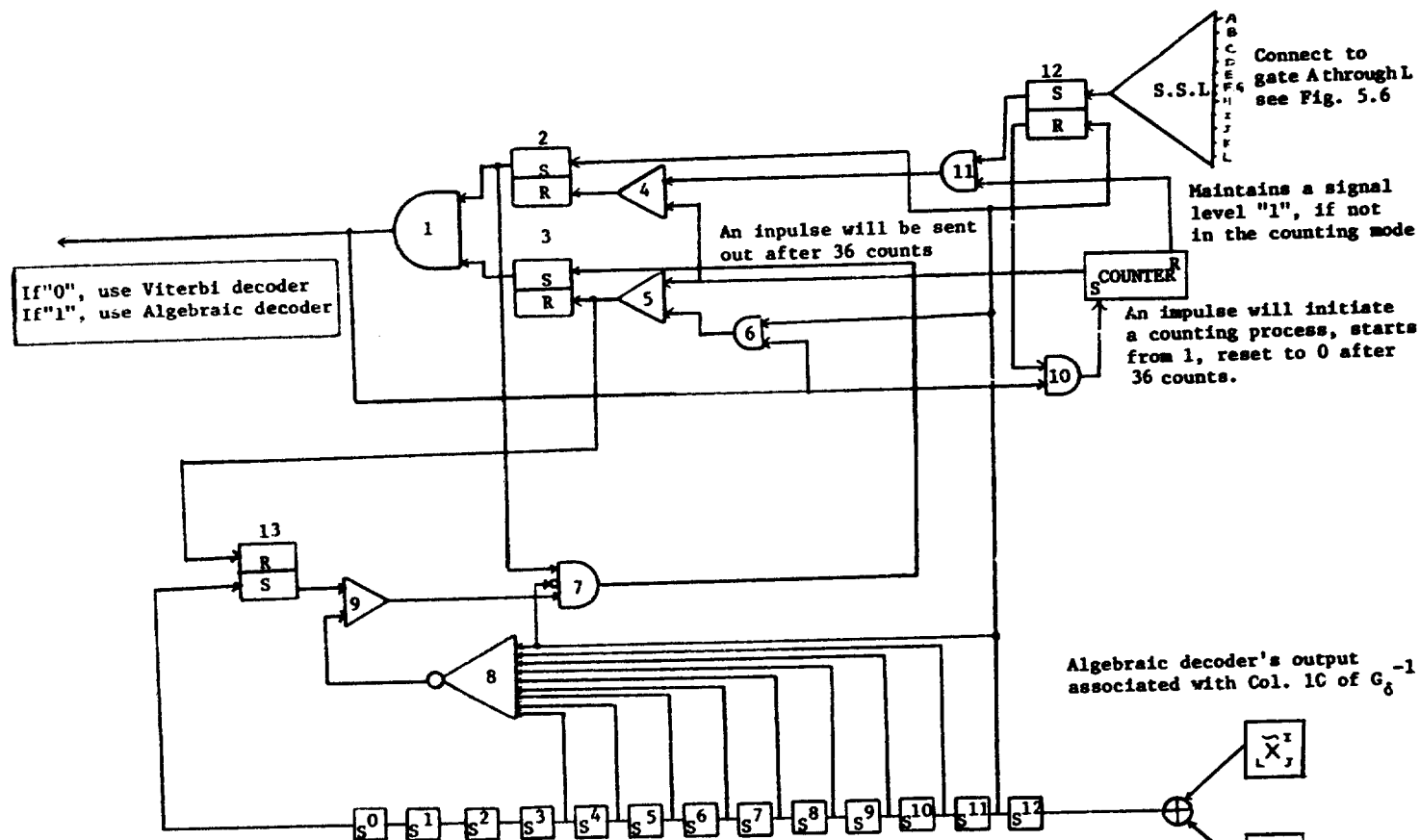
In the normal situation, the channel is supposed to be clean and no action shall be taken by the S.D.L.; the output of gate 1 is always "0", which means the decoding duty is assumed by the Viterbi decoder. However, in case of any error event, a "1" will enter the shift register stages within 12 operations from the occurrence of the event. As soon as the "1" enters the shift register, gate 2 will be actuated and the S.D.L. is in a stand-by position. Should the event be a random error or a short burst, gate 2 will be reset to its original state as soon as the first "1" in the syndrome reaches S^3 and the alert is released. If the event is not a random error or short burst, gate 2 will stay in the "set" position until the first "1" in the syndrome reaches S^3 , i.e. a 12 bit delay since the first

appearance of the nonzero syndrome bit, or 9 consecutive zeros have entered the shift register, whichever happens first. And conditioned on the fact that gate 2 is still in the "set" position, i.e. the event is not a random error or a short burst, then gate 3 will be actuated. With both gate 2 and gate 3 in the "set" position, gate 1 will produce a "1" output which means the decoding action shall be shifted to the algebraic decoder. As soon as the decoding duty is assumed by the algebraic decoder, a counter starts to count. At the end of a 36 count, both gates 2 and 3 will be turned off and the S.D.L. returned to its normal state. However, if the channel encounters another error prior to the ending of the 36 counts, the decoding will also be returned to the Viterbi decoder, but the S.D.L. will remain in the "alert" mode. Should the second event be a random error or short burst, the S.D.L. will return to its normal state as usual; otherwise, the decoding duty will be assigned to the algebraic decoder after the burst is over, in the same manner as if this event were encountered for the first time. Therefore, the S.D.L. switches the decoding actions back and forth between the two decoders according to a set of predetermined decision rules.

The only drawback of this algorithm is that when the channel encounters a short burst while the decoding is in the algebraic mode, it will never go back to the algebraic decoder after the burst is over, even if the Viterbi decoder is not re-synchronized, i.e. the counter has not finished the 36 counts. This problem can be circumvented by an "AND" at the output of S.S.L. with the complement of the counter's output so that if the counter is in the counting mode,

gate 2 cannot be reset, i.e. gate 2 can be turned off if, and only if, the counter is not in the counting mode. Furthermore, the feedback connection between the output end of gate 5 and the counter shall be removed, which means that once the counter starts counting, nothing can terminate it until it finishes the 36 counts. This alternative implementation also has its shortcomings. It regards all the short bursts which occur when the counter is in the counting mode as long bursts and re-initiates the counter with each new occurrence of a burst. If the channel is rich in random errors or short bursts, the decoding will return to the Viterbi decoder momentarily and stay with algebraic decoder the rest of the time. Methods must be incorporated so that the counter can be initiated only when the burst is long. This can be done by conditioning the gate 1's output with the S.S.L. output before sending to the "SET" terminal of the counter, so that the counter cannot be re-initiated unless the S.S.L. output is "0", i.e. the event is a long burst. With such modifications, every time the hybrid decoder encounters a long burst, the next 36 decoded bits are regarded as unreliable regardless of whether the occurrence is within the 36 bits span or not; however, the occurrence of random errors or short bursts will not initiate the counter, even if the hybrid system is in the counting mode; the only action that the system takes is to return the decoding to the Viterbi decoder for as long as the error event lasts. For the details of this modified S.D.L., please see Fig. 5.4.

Since during the counting period the Viterbi decoder's outputs are not reliable, temporarily returning the decoding operations to



Note: If the channel encounters a second error while the hybrid decoder is in the counting mode, decoding will be returned to the Viterbi decoder temporarily, then goes back to the Viterbi decoder to finish the 36 counts. If the second error is a long burst, starts counting from zero again.

Figure 5.4 Syndrome Detecting Logic

this decoder does not guarantee error free results. This fact leads to the consideration of a third variation. It is known that the algebraic decoder is not just a syndrome creator, but can also serve as a multiple parity check decoder for convolutional codes. The only requirement is that $\delta = 6$ blocks of error free guard space must be provided. This is true in general with very high probability if random errors are the case. Under the condition that the channel is indeed so benevolent, the algebraic decoder may be used to correct all errors when the Viterbi decoder is out of synchronization. The algebraic decoder is capable of correcting all error patterns that are limited to within 2 blocks, i.e. 4 bits, which is exactly the same as that of the Viterbi decoder for short blocks. Theoretically, the error correcting potential of the algebraic decoder is up to bursts of 6 blocks long, but the complexity of the hardware or the memory storage required forbids such implementation. Even if the error correcting of the algebraic decoder is limited to random errors and short bursts, the third variation would perform much better than the first two if the channel is known to give enough error free guard space. The choice is actually dependent, not upon the overall error rate but upon the channel characteristics, that is whether the channel has more random errors or bursty errors, or whether the channel error events are sparse enough to provide clean guard space. The S.D.L. for this variation is shown in Fig. 5.5 and Fig. 5.6.

5.3 Computer Simulation

The hybrid decoding process developed, together with the Viterbi and algebraic decoders, was simulated on a Univac 1108 to verify its

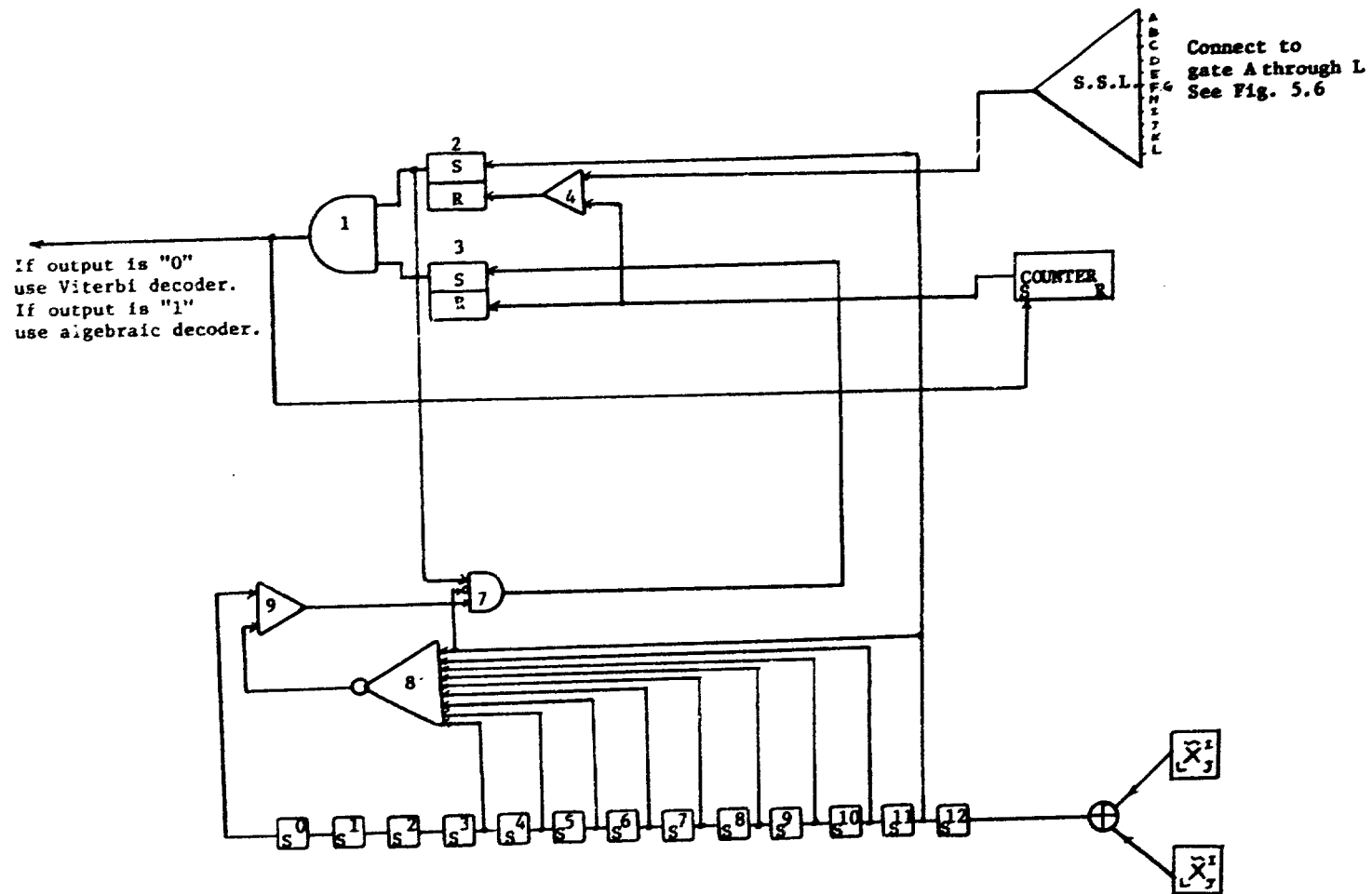


Figure 5.5 Syndrome Detecting Logic

S.S.L. Inputs Connection Table

input gate	s^1	s^2	s^3	s^4	s^5	s^6	s^7	s^8	s^9	s^{10}	s^{11}	s^{12}
A	1	0	1	1	0	1	1	0	0	0	0	0
B	1	1	1	1	0	0	1	0	0	0	0	0
C	1	1	1	0	1	1	0	1	0	0	0	0
D	1	0	1	0	1	0	0	1	0	0	0	0
E	1	1	0	0	1	1	1	1	0	0	0	0
F,G	1	0	0	0	1	0	1	-	0	0	0	0
H	1	1	0	1	0	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0	0	0	0
J	1	1	1	1	1	0	0	0	0	-	-	-
K	1	1	1	1	0	1	0	0	0	0	-	-
L	1	1	0	0	1	1	0	0	0	0	0	-

Note:
The "AND" gate shown is a typical gate that feeds into the "OR" gate labelled as **S.S.L.**. The inputs of the "AND" gate are connected to the shift register stages s^i , $i = 1, 2, \dots, 12$. Whether these s^i outputs shall be complemented or not before feeding into the corresponding "AND" gate, please see the "S.S.L. Inputs Connection Table". If the corresponding entry in the table is a "1" do not complement it, if a "0", complement it, if a "-" is shown, which means "don't care", do not input that stage at all.

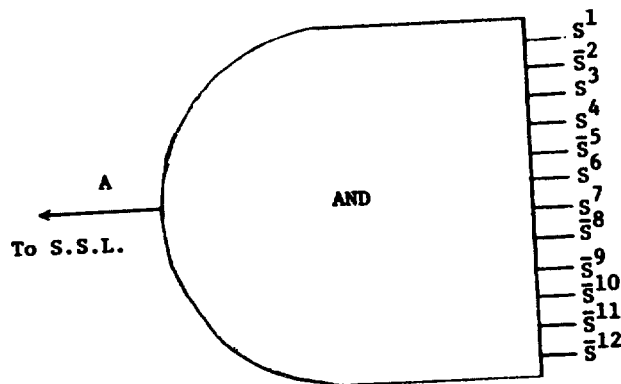


Figure 5.6 Syndrome Search Logic (S.S.L.) Connection Diagram

error correcting capability and to compare the performances of the three. Details of the simulation are given in the Appendix. Random information was encoded using the $(2,1)$, constraint length 7 convolutional code.

Twenty-three complex channels were simulated and 2000 words, or 64000 bits, were used for each test run. For each channel the nominal error rate was made 1%, 0.1% and 0.01%. Channels were characterized by both random errors and bursts with burst lengths ranging from 1 block to 6 blocks. The 23 channels ranged from channels with 0% random errors, i.e. pure bursty channels to channels with 100% random errors, with an increment of 5% random errors for each successive simulation. The frequency of bursty events was always made inversely proportional to the burst lengths. Whenever a burst was simulated, a random number truncated to the appropriate length was inserted to the error sequence; however, for a random error only the corresponding received bit was inverted. The two additional channels simulated were the channel with strictly short bursts and the channel with strictly long bursts.

A syndrome table consisting of all the syndrome patterns for short bursts was stored so that the decoding of short bursts could be made definite. Long bursts were decoded according to the standard method discussed earlier, that is, identifying the correct decoding operation and using the outputs of that operation to replace all the previously decoded bits. Thus, a finite chance of erroneous decoding is possible.

A buffer register of length equivalent to the decoder's constraint length of the Viterbi decoder was used to temporarily hold

the algebraic decoder's output; the function of this buffer is two-fold, first, for holding the algebraic decoder's outputs for later retrieval and possible corrections, second, for synchronization with the Viterbi decoder's output. After an initial DCL bits delay, the syndrome detecting logic commanded the system to accept one of the sub-decoder's output according to a bookkeeping register's content-- if "1" was indicated, using the algebraic decoder's output, otherwise using Viterbi's output.

Parameters monitored were error counts and error probabilities for each type of error event and in addition, the number of errors per word; each word was regarded as 32 bits for the code tested. All were recorded before decoding and after decoding and for all three decoders. Finally, an overall performance measure was calculated in terms of both error reduction and error ratio.

5.4 Experimental Results and Findings

Table 5.2 summarizes the results of all the 23 simulations. It can be easily seen that the hybrid decoder out-performed both the Viterbi decoder and the algebraic decoder in every case. The only exception was that for the 1% random error channel, the hybrid decoder had one decoding error while the Viterbi decoder corrected all the errors. This could be an accident, however, it is also anticipated that while the hybrid decoder is the best decoder for compound channels, its performance can only approach that of the Viterbi decoder in strict memoryless channels, but can never exceed it.

TABLE 5.2

COMPARISON OF CHANNEL PERFORMANCE FOR DIFFERENT CODING SCHEMES

% OF RANDOM ERROR	NOMINAL CHANNEL ERROR PROBABILITY PERFORMANCE	1%		0.1%		0.01%	
		NO. OF ERROR BITS	CHANNEL IMPROVEMENT = $\frac{\text{E.P. with Coding}}{\text{E.P. without coding}}$	NO. OF ERROR BITS	CHANNEL IMPROVEMENT = $\frac{\text{E.P. with Coding}}{\text{E.P. without Coding}}$	NO. OF ERROR BITS	CHANNEL IMPROVEMENT = $\frac{\text{E.P. with Coding}}{\text{E.P. without Coding}}$
0%	RECEIVED SEQUENCE	3840		493		22	
	DECODED SEQUENCE: VITERBI	543	0.282812	53	0.215010	0	
	ALGEBRAIC	631	0.328646	41	0.166329	0	
	HYBRID	325	0.169271	32	0.129817	0	
	RATIO OF IMPROVEMENT: HYBRID/VITERBI		0.598528		0.603772		
5%	RECEIVED SEQUENCE	3727		462		21	
	DECODED SEQUENCE: VITERBI	527	0.282801	37	0.160173	0	
	ALGEBRAIC	670	0.359538	46	0.199134	0	
	HYBRID	325	0.174403	26	0.112554	0	
	RATIO OF IMPROVEMENT		0.616699		0.702703		
10%	RECEIVED SEQUENCE	3632		442		20	
	DECODED SEQUENCE: VITERBI	519	0.285793	27	0.122172	0	
	ALGEBRAIC	680	0.374449	42	0.190045	0	
	HYBRID	348	0.191630	17	0.076923	0	
	RATIO OF IMPROVEMENT		0.670520		0.629629	0	
15%	RECEIVED SEQUENCE	3513		421		19	
	DECODED SEQUENCE: VITERBI	460	0.261884	23	0.109264	0	
	ALGEBRAIC	679	0.386564	26	0.123515	0	
	HYBRID	300	0.170794	13	0.061758	0	
	RATIO OF IMPROVEMENT		0.652174		0.565218		
20%	RECEIVED SEQUENCE	3419		400		16	
	DECODED SEQUENCE: VITERBI	459	0.268500	16	0.080000	0	
	ALGEBRAIC	676	0.395437	19	0.095000	0	
	HYBRID	293	0.171395	6	0.030000	0	
	RATIO OF IMPROVEMENT		0.638343		0.375000		

TABLE 5.2
(Continued)

COMPARISON OF CHANNEL PERFORMANCE FOR DIFFERENT CODING SCHEMES

Z OF RANDOM ERROR	NOMINAL CHANNEL ERROR PROBABILITY PERFORMANCE	1%		0.1%		0.01%	
		NO. OF ERROR BITS	CHANNEL IMPROVEMENT E.P. with Coding E.P. without Coding	NO. OF ERROR BITS	CHANNEL IMPROVEMENT E.P. with Coding E.P. without Coding	NO. OF ERROR BITS	CHANNEL IMPROVEMENT E.P. with Coding E.P. without Coding
25Z	RECEIVED SEQUENCE	3330		385		15	
	DECODED SEQUENCE: VITERBI	432	0.259459	27	0.140260	0	
	ALGEBRAIC	691	0.115015	21	0.109091	0	
	HYBRID	277	0.166366	16	0.083117	0	
	RATIO OF IMPROVEMENT: HYBRID/VITERBI		0.641203		0.592592		
30Z	RECEIVED SEQUENCE	3231		366		13	
	DECODED SEQUENCE: VITERBI	418	0.258743	17	0.092896	0	
	ALGEBRAIC	658	0.407303	25	0.136612	0	
	HYBRID	255	0.157846	10	0.054645	0	
	RATIO OF IMPROVEMENT		0.610049		0.588238		
35Z	RECEIVED SEQUENCE	3111		341		12	
	DECODED SEQUENCE: VITERBI	345	0.221794	8	0.046921	0	
	ALGEBRAIC	674	0.433301	21	0.123167	0	
	HYBRID	219	0.140791	5	0.029326	0	
	RATIO OF IMPROVEMENT		0.634783		0.625008		
40Z	RECEIVED SEQUENCE	3017		333		11	
	DECODED SEQUENCE: VITERBI	326	0.216109	10	0.060060	0	
	ALGEBRAIC	650	0.430892	22	0.132132	0	
	HYBRID	204	0.135234	5	0.030030	0	
	RATIO OF IMPROVEMENT		0.625768		0.500000		
45Z	RECEIVED SEQUENCE	2905		310		10	
	DECODED SEQUENCE: VITERBI	329	0.226506	9	0.058065	0	
	ALGEBRAIC	662	0.455766	25	0.161290	0	
	HYBRID	220	0.151463	6	0.038710	0	
	RATIO OF IMPROVEMENT		0.668693		0.666667		

ORIGINAL PAGE IS
OF POOR QUALITY

TABLE 5. 2

(Continued)

COMPARISON OF CHANNEL PERFORMANCE FOR DIFFERENT CODING SCHEMES

2 OF RANDOM ERROR	NOMINAL CHANNEL ERROR PROBABILITY PERFORMANCE	1Z		0.1Z		0.01Z	
		NO. OF ERROR BITS	CHANNEL IMPROVEMENT E.P. with Coding E.P. without Coding	NO. OF ERROR BITS	CHANNEL IMPROVEMENT E.P. with Coding E.P. without Coding	NO. OF ERROR BITS	CHANNEL IMPROVEMENT E.P. with coding E.P. without coding
1/2	RECEIVED SEQUENCE	2793		286		10	
	DECODED SEQUENCE: VITERBI	308	0.220551	7	0.048951	0	
	ALGEBRAIC	696	0.492389	24	0.167832	0	
	HYBRID	212	0.151208	6	0.041958	0	
	RATIO OF IMPROVEMENT: HYBRID/VITERBI		0.688312		0.857143		
1/3	RECEIVED SEQUENCE	2652		269		9	
	DECODED SEQUENCE: VITERBI	227	0.171192	7	0.052045	0	
	ALGEBRAIC	693	0.522624	20	0.148499	0	
	HYBRID	172	0.129713	6	0.044610	0	
	RATIO OF IMPROVEMENT		0.757795		0.857143	0	
1/6	RECEIVED SEQUENCE	2511		250		9	
	DECODED SEQUENCE: VITERBI	219	0.174432	7	0.056000	0	
	ALGEBRAIC	756	0.602151	33	0.264000	0	
	HYBRID	187	0.148945	6	0.048000	0	
	RATIO OF IMPROVEMENT		0.853886		0.857143		
1/12	RECEIVED SEQUENCE	2371		236		9	
	DECODED SEQUENCE: VITERBI	192	0.161957	3	0.025424	0	
	ALGEBRAIC	672	0.566849	19	0.161017	0	
	HYBRID	131	0.110592	3	0.025425	0	
	RATIO OF IMPROVEMENT		0.642292		1.000000		
7/12	RECEIVED SEQUENCE	2241		212		9	
	DECODED SEQUENCE: VITERBI	166	0.148148	5	0.047170	0	
	ALGEBRAIC	696	0.621151	5	0.047170	0	
	HYBRID	126	0.112450	5	0.047170	0	
	RATIO OF IMPROVEMENT		0.759038				

TABLE 5. 2

(Continued)

COMPARISON OF CHANNEL PERFORMANCE FOR DIFFERENT CODING SCHEMES

X OF RANDOM ERROR	NOMINAL CHANNEL ERROR PERFORMANCE	1%		0.1%		0.01%	
		NO. OF ERROR BITS	CHANNEL IMPROVEMENT E.P. with Coding E.P. without Coding	NO. OF ERROR	CHANNEL IMPROVEMENT E.P. with Coding E.P. without Coding	NO. OF ERROR BITS	CHANNEL IMPROVEMENT E.P. with coding E.P. without Coding
75%	RECEIVED SEQUENCE	2087		194		9	
	DECODED SEQUENCE: VITERBI	113	0.108289	0	0.000000	0	
	ALGEBRAIC	609	0.583613	17	0.175258	0	
	HYBRID	75	0.071874	0	0.000000	0	
	RATIO OF IMPROVEMENT: HYBRID/VITERBI		0.663724				
80%	RECEIVED SEQUENCE	1942		177		9	
	DECODED SEQUENCE: VITERBI	117	0.120494	2	0.022599	0	
	ALGEBRAIC	634	0.652935	12	0.135593	0	
	HYBRID	75	0.077240	1	0.011299	0	
	RATIO OF IMPROVEMENT		0.641028		0.499978		
85%	RECEIVED SEQUENCE	1778		160		8	
	DECODED SEQUENCE: VITERBI	85	0.095613	0	0.000000	0	
	ALGEBRAIC	655	0.736783	11	0.137500	0	
	HYBRID	48	0.053993	0	0.000000	0	
	RATIO OF IMPROVEMENT		0.564704				
90%	RECEIVED SEQUENCE	1639		149		8	
	DECODED SEQUENCE: VITERBI	62	0.075656	0	0.0	0	
	ALGEBRAIC	660	0.805369	8	0.107383	0	
	HYBRID	48	0.058572	0	0.0	0	
	RATIO OF IMPROVEMENT		0.774188				
95%	RECEIVED SEQUENCE	1452		137		8	
	DECODED SEQUENCE: VITERBI	12	0.016529	0	0.000000	0	
	ALGEBRAIC	645	0.888430	7	0.102190	0	
	HYBRID	12	0.016529	0	0.000000	0	
	RATIO OF IMPROVEMENT		1.000000				

ORIGINAL PAGE IS
OF POOR QUALITY

TABLE 5.2
(Continued)
COMPARISON OF CHANNEL PERFORMANCE FOR DIFFERENT CODING SCHEMES

Z OF RANDOM ERROR	NOMINAL CHANNEL ERROR PROBABILITY PERFORMANCE	1%		0.1%		0.01%	
		NO. OF ERROR BITS	CHANNEL IMPROVEMENT	NO. OF ERROR BITS	CHANNEL IMPROVEMENT	NO. OF ERROR BITS	CHANNEL IMPROVEMENT
			<u>E.P. with Coding</u> E.P. without Coding		<u>E.P. with Coding</u> E.P. without Coding		<u>E.P. with Coding</u> E.P. without Coding
100%	RECEIVED SEQUENCE	1131		135		8	
	DECODED SEQUENCE: VITERBI	0	0.000000	0	0.000000	0	
	ALGEBRAIC	632	0.964150	7	0.103704	0	
	HYBRID	2	0.003051	0	0.000000	0	
	RATIO OF IMPROVEMENT: HYBRID/VITERBI						
SHORT BURST ONLY	RECEIVED SEQUENCE	2487		310		16	
	DECODED SEQUENCE: VITERBI	11	0.008846	0	0.000000	0	
	ALGEBRAIC	0	0.000000	0	0.000000	0	
	HYBRID	0	0.000000	0	0.000000	0	
	RATIO OF IMPROVEMENT						
LONG BURST ONLY	RECEIVED SEQUENCE	4911		643		31	
	DECODED SEQUENCE: VITERBI	1225	0.498880	72	0.223950	1	0.064516
	ALGEBRAIC	1183	0.481776	87	0.270607	2	0.129032
	HYBRID	732	0.298106	43	0.133748	1	0.064516
	RATIO OF IMPROVEMENT		0.597551		0.597223		1.000000

Due to the small amount of data tested, i.e. 64000 bits for each test, the statistics obtained did not yet seem stabilized, especially for those low noise channels. However, the collected data did reveal the tendency of improving performance with increasing burst contents of the channels. This phenomenon was even more pronounced for low noise channels. As an extreme case, the short burst only channel had all errors removed after the hybrid decoding but had 11 error bits left when Viterbi decoder was applied; the overall performance gain of the hybrid decoder over the Viterbi decoder was around 40% to 50%. This is a tremendous improvement and there is no known algorithm which can perform so well, therefore, the goal of achieving an algorithm whose performance approaches that of the maximum likelihood Viterbi decoder in combatting random errors and yet possesses a burst error correcting capability approaching that of the optimal B2 burst corrector was "almost" fulfilled. The reason the word "almost" is used is that the hybrid algorithm does not exactly match the theoretical performances of the two. Further discussion about this point follows in the next section; it is hoped that some light can be shed on further improvements of this algorithm.

5.5 Discussion and Conclusion

The main drawback of the proposed hybrid decoder is the inability of the syndrome detecting logic to tell the beginning and the ending of an error event definitely, although it could tell with high probability. The second drawback is that the syndrome detecting logic could not distinguish a short burst from a random error; all random

errors were treated as short bursts. This may not be a hazard if the random errors are sparsely distributed. However, if the successive random errors are not separated by enough clean guard space, it will be mistaken as a long burst and allocated to the algebraic decoder. Clearly, the algebraic decoder may commit more mistakes than the Viterbi decoder for such "pseudo-bursts", and thus become a source of deterioration for the hybrid decoder. However, this deterioration would not happen if the syndrome detecting logic could correctly identify the beginning and the ending of an error event. Therefore, the first drawback seems the most serious one.

Another factor which has a decisive influence on the performance of the hybrid decoder is the number of consecutive zeros in the syndrome patterns. If the maximum number of consecutive zeros over all syndrome patterns were small, the decision of the event "error is over" could be made much earlier, then the guard space requirement could be significantly relaxed, and many long bursts could be broken up into several short bursts. If this were true, the overall performance of the hybrid decoder would also be improved.

So far all the drawbacks of the proposed hybrid decoder have been mentioned. These drawbacks are difficult to remove. First, it is not possible to store all the syndrome patterns because of storage space and the searching time involved. Second, the number of syndrome patterns are in the same order as the number of error patterns and are of the same length. For distinct syndrome patterns to exist, about $3/4$ of the n -tuples in the n -space must be selected. This will not allow all the syndrome patterns to have a "1" in the first position and a "1" in the last position, and the all-zero blocks within all

patterns will be very short. Unless some other approach is taken, no drastic improvements can be seen.

The proposed hybrid decoder, though it has these drawbacks, is in performance still superior to any known algorithms for the compound channel. This is because that the chances of encountering a "bad" syndrome pattern are low and the "bad" syndrome pattern causing an erroneous decoding is still lower. Even for arbitrarily long bursts, the algorithm allows the decoding errors committed by the algebraic decoder to be limited to within $(\Gamma + \delta)l$ bits and the rest of the sequence is passed to the Viterbi decoder. Furthermore, the error propagation effect of the Viterbi decoder is also suppressed. Error propagation suppression can be achieved for any error event, regardless of whether it can be corrected by the algebraic decoder or not; therefore, this is the biggest advantage of the hybrid decoder and one of the factors that is responsible for the overall improvement of the algorithm.

Based on the above observation, it seems that the proposed algorithm works best when the channel is rich in short bursts and worse when the channel is strictly random and the random errors are not separated by enough clean guard spaces. Since purely random events, by definition, must be distributed evenly over their sample space and not tend to cluster together, for channels with reasonable error density clean guard spaces should be achieved. Therefore, it is those "nearly random and nearly bursty errors" that cause troubles. Just where the dividing line is between "a group of random errors" and "one long burst" is not quite clear. The correct defining of the two kinds of errors may have to depend on a better understanding of

the error correcting capability of the Viterbi algorithm. Those long bursts with error density low enough to allow correct decoding by the Viterbi decoder may be defined as random errors; otherwise, they would be a true burst. This, of course, is a definition from the practical point of view, not necessarily having any theoretical significance.

The number of consecutive zeros that has to be checked in the syndrome sequence before a decision is made depends on the maximum length of the all-zero sequence within all syndrome patterns available. However, if this length is too long, a little compromise may have to be made to assume a shorter length. This is true because if the decision is made too late, the ending of some of the shorter bursts may be overlooked and several adjacent shorter bursts may be regarded as a long burst. The practical choice of the length of "zero check" is actually a trial-and-error process; it depends on the syndrome patterns as well as the channel error distribution. The optimal choice of the length may be determined empirically through computer simulation; however, such determination is time and money consuming, and it varies from channel to channel. The reason that 9 has been used as the figure is that apparently this is the maximum consecutive zeros for most syndrome patterns for the $(2,1)$, $k = 7$ code.

CHAPTER VI

RECOMMENDATIONS

6.1 The Goal

Theoretical formulation indicated that the rate $1/2$, constraint length 7 code used for simulation should be an optimal B2 burst corrector. Its random error correcting capability is well-known to be very good--actually the best code that has ever been used in practical applications. This code is capable of correcting 6 consecutive blocks of errors in a 12 block span; that is, even if half of the received sequence is in error, no information has been lost as long as the erroneous bit streams are separated by clean guard spaces of 6 blocks or longer. For error events that are not followed by these clean guard spaces, the information content is still recoverable if they are randomly distributed with a short length of 1 to 2 blocks as these are deemed correctable by the Viterbi decoder from an earlier study. The goal is to find a decoding algorithm which can fully exploit the potential burst-correcting capability of the code and yet perform as well as the Viterbi decoder in dealing with the random errors.

6.2 Problems and Possible Approaches of Attacking These Problems

As mentioned in Sec. 5.5, the proposed algorithm performs well for compound channels but it has not reached the full error correcting

potential of the code. Several immediate problems encountered during the realization of the proposed hybrid algorithm were also pointed out in that section. Here it is well to reconsider these problems:

1. The syndrome patterns do not always begin with a "1" and end with a "1". The damages caused by this fact might not be when the errors are segmented by long clean guard spaces but rather when two consecutive bursts, the syndrome of one prefixed with several zeros and the other suffixed with several zeros, are so close that the zeros of both syndromes are overlapping. Since the chances of such events are very low, they may not be a serious problem. One possible solution might be the use of "syndrome removal" instead of "syndrome resetting". After each decoding operation, methods may be used to calculate the syndrome of the assumed error vector and subtract it from the content of the syndrome register. If the assumption made is correct, the syndrome register will be zero; otherwise, the correction is improper and it should be attempted again at the next operation. It would be even better if methods could be devised to extract information about the error vector from the remainder of the syndrome register after each syndrome removal. Another approach is to re-encode the decoded sequence. If both sequences are in perfect agreement, it may be assumed that the received sequence is error free and this fact may be used to check the clean guard space requirement. However, the above statement is only a necessary

condition--not a sufficient one--for being "error-free".

This is true because the re-encoded sequence and the received sequence are still in perfect agreement, provided the received sequence is not "error-free" and no correction has been made. As a matter of fact, these two sequences are always in agreement with each other if there has been no correction attempted. Therefore this method is deemed not usable for the hybrid system.

2. The number of consecutive zeros within the syndrome patterns causes a delay in recognition of the ending of an error event. Theoretically two consecutive bursts may not show any gap between their syndrome sequences if they are space with a clean guard space of minimum length, i.e. 6 blocks for the example code. However, the syndrome patterns are not the all "1" vectors, as a matter of fact, there may be several consecutive zeros within a syndrome pattern; therefore, it cannot be concluded that the error is over by examining one "0" in the syndrome sequence, not even after seeing several "0"s. What can be done is to examine all the possible syndrome patterns and to find out just what the maximum number of consecutive "0"s among all the syndrome patterns is. However if this number plus 1 is used as the decision-making timing, the ending of some bursts may be overlooked if they are followed "too close" by other bursts. This may result in combining several shorter bursts into a long burst with some of the error-correcting capability of the algebraic decoder

sacrificed. If the channel error distribution is such that the error events are sparse, the above-mentioned method will not cause significant deterioration, otherwise extra caution must be exercised. One empirical solution is to perform experiments, using various "zero-check" lengths, to determine which one gives the best performance.

3. The syndrome detecting logic used for the proposed hybrid decoder can only differentiate short bursts from long bursts. Random errors may be treated as either short bursts or long bursts, depending on when the error sequence obtained the required clean guard space. If a group of random errors is treated as a short burst, no problem will result. However, if it is treated as a long burst, the nondefinite decoding of long bursts by the algebraic decoder and the excellent random-error-correcting capability of the Viterbi decoder would, on the average, cause deterioration. Just what could be done for this problem is not quite clear at the moment. This is true because little is known about the Viterbi decoder's performance with respect to the density of error bits in the received sequence.

It is felt that full information content of a contaminated sequence must be extractable if the "complexity of hardwares" is disregarded. Probably a completely different approach should be taken in the forming of syndrome patterns and the designing of syndrome search logic. It is also possible that the proposed algorithm is the optimal within realizability. Much is left to be determined by future

investigations. A more ambitious task is the studying of the structure properties of the convolution codes by both the finite-state machine and the algebraic point of views and the inter-relation between the two approaches. In any case, it is felt that a hybrid system using both properties must be used in order to exploit fully the burst-error-correcting capabilities of the convolutional codes. This is because the optimal random error correcting decoder is already available, i.e. the Viterbi decoder.

6.3 Further Improvement of the Proposed Algorithm

If a little bit more "complexity" disadvantage can be tolerated, further improvement of the proposed algorithm is still possible. Since the reduction of the number of errors per words is more pronounced than the reduction of total errors, it is possible to concatenate the proposed algorithm with a block code, for instance a triple error correcting BCH code, to further remove the remaining errors. If the application requires stronger burst correction, then interlacing may be used prior to hybrid decoding. If high performance is required for both random errors reduction and bursts reduction, then both concatenation and interlacing may be used. Since the proposed algorithm uses only one code with two decoding algorithms applied alternatively, additional concatenation and interlacing will not complicate the overall system beyond toleration.

APPENDIX

SIMULATION

As pointed out in Sec. 5.2.3, there are many different ways of interfacing the two algorithms and each has its advantages and disadvantages. It was found during actual computer simulations that the last variation, with some additional modifications, worked the best among others. Simulations were performed on Univac 1108 multiprocessor. The final procedure adapted was to let the algebraic decoder correct all the bursts, long and short. Random errors were treated as short bursts if they were followed by enough clean guard space; otherwise, they were passed to the Viterbi decoder to be decoded probabilistically. This is because the algebraic decoder can correct short bursts definitely while the Viterbi decoder can only decode probabilistically.

Both source data and channel statistics were created by using the pseudo-random numbers. The general process was to create source words, each consisting of 32 bits, encoding these words, adding channel errors in a fashion consistent with specified channel statistics, and then decoding the received sequences using differing algorithms. Parameters were monitored which allowed calculation of probability of bit error rates for both coded and uncoded systems. Fig. A.3 is a sample of the output format which shows the type of data monitored, collected, and tabulated. Table A.1 lists the input parameters that have to be specified.

The computer program developed allows the simulation of any rate $1/n$ code, as long as the constraint length of the encoder is less than 32 stages; however, some minor adjustments must be made if $n \neq 2$ and the generator matrix is not of size 12×12 .

APPENDIX A

ENCODER

Encoding is performed sequentially in strict accordance with the diagrams shown in Fig. 2.4 or Fig. 4.4. Each encoding operation causes one bit of the source sequence entering the shift register. Then the contents of the corresponding stages are mod-2 added according to the connection vectors. After the outputs of these EXCLUSIVE ORs were transferred to the codeword register, the contents of the shift register were shifted sequentially one place to the right. Then the shift register was ready to receive the next input bit.

The source words were generated from a set of uniformly distributed pseudo-random numbers by referring to the seed number. The seed number was changed every time when a new set was requested to prevent repetitious data. Since convolutional codes are linear codes, the decoder's performance should be invariant to any message sent. This implies that an all zero sequence could be used to test the decoder's capability. Reasons for using pseudo-random numbers as test data are (1) to detect any non-linearities which might occur in the hybrid decoding algorithm and (2) for aesthetics.

APPENDIX B

CHANNEL

Channels of various statistics can be simulated by proper choice of the channel parameters. Determination of an error event is accomplished by examining a sequence of pseudo-random numbers uniformly distributed between 0 and 1. Since the probability of observing a number lying between a certain interval is simply the length of that interval, error events of various probabilities can be created by assigning proper length of intervals to each of them. For instances, if a 30% channel is desired; among the 30% errors, 15% are burst errors and among the burst errors, 10% are short bursts and 5% are long bursts, then the assignment is shown as follows:

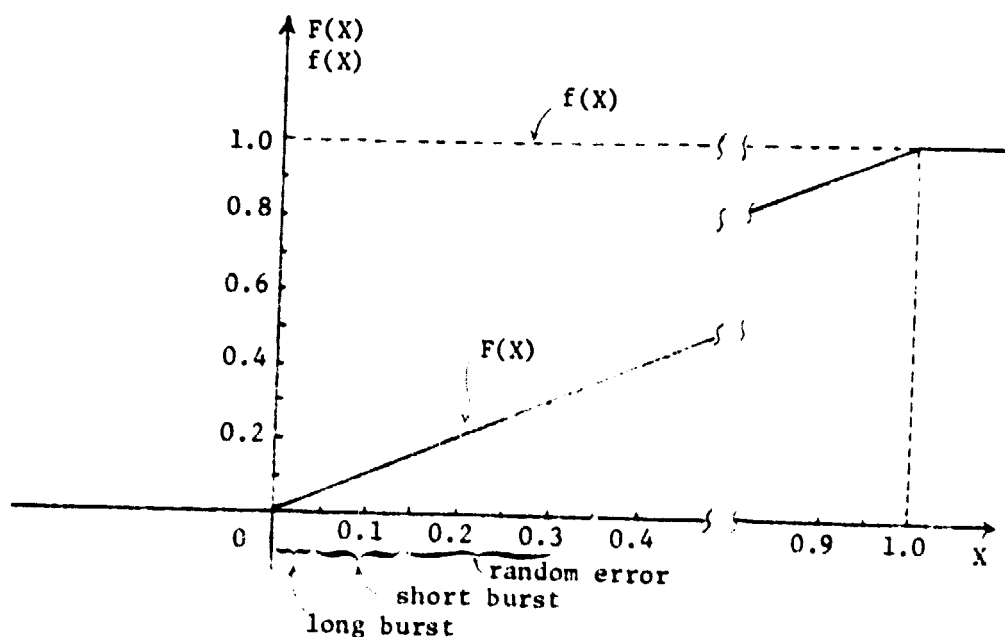


Fig A.1 Probability distribution function of the pseudo-random numbers.

142

Channel characteristics are simulated via an error sequence which is initially assumed to be all zero. The pseudo-random numbers are examined one by one. If a number in the random error interval is observed, the corresponding position of the error sequence is inverted into 1. However, if the number falls into the range of a burst interval, another random number is summoned, and its binary representation is truncated to the length of the corresponding burst, then stored in the proper positions of the error sequence. This is done because the bursts should not be limited to the all "1" pattern. With this statistically created error sequence, channel characteristics are fully described. Transmission of the message sequence over the channel is simply a modulo-2 addition of this sequence and the error sequence; the output sequence is stored in the received word register to be decoded.

The simulations performed in this experiment are not intended to model any particular channel but rather to create the gross statistics of most channels. Of course, the program developed is capable of modeling any particular channel if parameters describing the channel are made available.

APPENDIX C

VITERBI DECODER

The Viterbi algorithm used in the program is exactly as described in Sec. 3.2.1.

First, a decoding table is constructed and stored. This table contains all the information of the trellis diagram; it provides output data for all possible states with all possible input combinations.

For each decoding operation, one received block is fetched from the received word register. Since there are 2^L paths leading to each node of the trellis, the outcomes of these paths are compared with the received block to determine which one is the most likely path; this has to be done for all nodes. The path that causes the least number of errors is selected for each state and the number of errors caused is recognized as the score of that state. At the same time, the corresponding input bit is stored in the survivor sequence register. The following decoding steps compare the 2^L paths leading to the same node, but the selection of the optimal path is based on the accumulated score of the previous operations and the current operation. Otherwise, the procedures are the same. The lengths of the survivor registers are made equivalent to the decoder's constraint length. Since the contents of these registers are shifted to the right one place after each operation, outputs start dumping out after DCL operations. Furthermore, all the survivor paths will

be merged together before reaching the end of the registers. The outputs pouring out from any of them are the same; customarily the first survivor register is used for collecting the decoded sequence. The reason that all the 2^{k-1} , i.e. the number of states, survivor registers must be kept is that during the course of decoding, their entire contents are needed for possible interchanges.

APPENDIX D

ALGEBRAIC DECODER AND SYNDROME DETECTING LOGIC

Algebraic decoding is achieved by postmultiplication of the received sequence with the inverse of the generator matrix, i.e. $\tilde{X}_s = r_s G_\delta^{-1}$. This is equivalent to modulo-2 addition of all the row vectors of G_δ whose corresponding multiplier bit is a "1". The process is as follows: shift the received sequence into the DA register, one block at a time; examine the content of DA bit by bit; at the same time advance the row vectors of G_δ^{-1} ; if a 1 is found in DA, retain that row vector; otherwise discard it; finally, modulo-2 add all the rows retained and store the result in XA2, the right most bit becomes the current decoded bit.

The syndrome bit of a particular operation is obtained by modulo-2 addition of the 10th bit of XA2 and XA1, counting from the left, where XA1 is simply the XA2 of the previous operation left-shifted one place. The syndrome bit obtained from each operation is stored sequentially in SYNRR to form the syndrome sequence.

Since the number of possible syndrome patterns grows exponentially with the length of the burst-error considered, only short bursts of 2 blocks long are stored in a subroutine called SYDSRH. The digital representation of these short bursts lies between 496 and 3872; therefore, if the content of SYNRR falls in this range, a search of the SYDSRH table starts. If the syndrome sequence matches exactly to one of the syndrome patterns tabulated, the ending and

the length of the error vector are known precisely and the decoding is definite. Since the nonzero interval of the syndrome patterns of the short burst varies from 5 to 8, the entire decoded sequences of 12 bits resulting from these steps of the operation are subscripted and stored for later retrieval.

Long bursts are those bursts which cannot be found in the SYDSRH table. The decoding of the long bursts depends on the examination of a certain number of consecutive zeros, designated as ZCK, in the syndrome sequence. If this criterion is satisfied, starting from the operation right ahead of the appearance of the first "0", counting back, 12 temporary decoded bits are replaced by the entire decoded sequence of that operation. It seems that the content of XA2 of every step has to be saved; however, this is not necessary. All that has to be done is to transfer the content of XA2 to a temporary register XALT at each step when the corresponding syndrome bit is not "0"; otherwise, no action shall be taken.

GLR register is used to keep track of the decoding actions of the algebraic decoder for the hybrid decoder's information. If an error event has been corrected by the decoder, a "1" is recorded in the corresponding position; otherwise the filling of a "0" or a "1" depends on whether that step of the operation for the corresponding hybrid system should be in mode 1 or mode 2, where mode 1 designates the Viterbi mode and mode 2, the algebraic mode.

Since an erroneous decoding of the Viterbi decoder may cause the outputs of the next 36 operations, i.e. 72 received bits, to be unreliable, during this period, the only useful results are those

coming from the algebraic decoder. Therefore, a counter CTR is employed to keep track of the out-of-synchronization period of the Viterbi decoder. If a long burst is detected, i.e. when the search of SYDSRH fails, as soon as the burst is over, CTR starts to count from 1, 2, ... up to 36, one increment after each decoding step; then it is reset to zero, waiting for the next initiation. During the counting period the hybrid system is regarded as in mode 2. Whenever the channel is clean, the corresponding bit position of the G1R register is assigned a "0", if the system is in mode 1; otherwise, "1" is assigned. Another function of the mode register is to remind the hybrid system to keep on counting, even if decoding is temporarily returned to the Viterbi decoder for some hybrid schemes, or for this particular scheme, when a short burst is encountered while in mode 2.

Since most times the error vectors are not as long as 12 bits, it is desirable to truncate the decoded vector in XALT to proper length before transfer and replacement; SYCC and ZCS counters are used for this purpose. SYCC tells the beginning and ZCS, the length of the replacement.

APPENDIX E

HYBRID DECODER

The hybrid decoder becomes very simple with its sub-decoders and the G1R register on hand. After an initial delay of 35 bits, all 3 registers, namely the decoded sequence register of both sub-decoders and the G1R register, start pouring out data. The hybrid decoder accepts outputs shifted out from either one of the decoded sequence registers according to the corresponding book-keeping bit of the G1R register. If a "0" is observed, the Viterbi decoder's output is accepted as the final decoded bit; otherwise the algebraic decoder's output is recognized. Since the hybrid decoding is performed parallel to its sub-decoders, bit-by-bit, no great time loss results. The whole system is operated in real time and outputs data uniformly. The only possible source that might cause nonuniform outputs is the searching of the syndrome table SYDSRH; therefore, the size of this table must be kept small.

Card 1

- NW - number of possible words
- NBPSW - number of bits per source word
- N - number of bits per source block
- K - number of EXCLUSIVE ORs
- CL - encoder constraint length
- DCL - decoder constraint length
- GDS - guard space to be inserted in the channel, could be entered as zero.
- ZCK - number of zeros in the syndrome sequence to be examined before making the decoding decision for long bursts.
- ASEED - the initial seed number to be assigned to induce a sequence of pseudo-random numbers, it could be any real number.

Card 2,3

- PBL and BL - BL denote the various error events and PBL are their corresponding probability assignments. They are sub-scribed variables; a total of 12 varieties can be accommodated; however, only minor changes of the program are required to increase this capacity.

Card 4

- HK1 and HK2 - the connection vectors of the encoder, represented in octal numbers. Can be increased if the number of EXCLUSIVE ORs of the encoder is more than 2.

Card 5,6

- G(I) - row vectors of the generator matrix, represented in octal number. G(I) is not used unless algebraic encoding is intended.
- G1(I) - row vectors of the inverse of the generator matrix, represented in octal number

If the generator matrix is other than 12 rows, minor adjustment must be made in the program.

TABLE A.1 Input parameters of the simulation program
-hybrid system.

```

COMPILE(FLOER)
DO 3000 MODEL,3
WRITE(6,1)
1 FORMAT(1H1,'CONVOLUTIONAL ENCODER---HYBRID DECODER WITH',/1X,'RA
  INCOM INPUT SEQUENCE AND RANDOM ERROR VECTOR',/1X,'')
  IMPLICIT INTEGER(I=0,R=2),REAL(A,P=0)
  DIMENSION SW(2002),DW(2002),DWV(2002),DWA(2002),DWH(2002),DER(2002)
  1)NEI(2002),NEO(2002),A(2000),AA(2000)
  DIMENSION CW(2002,2),EV(2002,2),RW(2002,2),CER(2002,2)
  DIMENSION TABLE(64,64),STATE(64),SKORE(64,64),SKORE(64),SSQ(64),SS
  1(64),SCORE(64)
  DIMENSION G(12),G1(12)
  DIMENSION DADS(8)
  DIMENSION BL(13),PBL(13),NB(13),PRBLE(13),NRLI(13),PRBLI(13),BECL(
  13),PBLI(13),PBLO(13),PRBLO(13),BECO(13),POBLO(13),NEPWI(13),PRPI
  2(13),NEPWO(13),PRBO(13),PB(13),PBP(13)
  DIMENSION BCL(13),POBLE(13)
  READ(5,3) NW,NBPSW,N,K,CL,BCL,GDS,ZCK,ASEED
  3 FORMAT(3I5,F10.2)
  READ(5,6) PCL
  6 FORMAT(8F10.8)
  READ(5,7) BL
  7 FORMAT(13I5)
  READ(5,10) NR1,NK2
  10 FORMAT(20I5)
  READ(5,12) (G(I),I=1,12)
  12 FORMAT(1205)
  READ(5,13) (G1(I),I=1,12)
  13 FORMAT(1205)
  COL=12
  ROW=12
  COL1=COL-1
  ROW1=ROW-1
  ZCK1=ZCK-1
  NR1=NR-1
  NK2=NK-2
  NR0=NR-6
  NR1=NR-1
  XN1=XN-1
  XN1=XN-1
  K1=K-1
  K2=K-2

```

Figure A.2 Simulation computer program-hybrid system

ORIGINAL PAGE IS
OF POOR QUALITY

C (b) CONVOLUTIONAL ENCODER

```

ASLE=ASELU+1
A(1)=ASEED
CALL RANDU(A,NW2)
SREG=0
KPT=K
DO 200 I=1,NW
  IF (I.GT.NW2) A(I)=0
  S(I)=NSWP*A(I)
  SWI=SW(I)
  DO 200 J=1,K
    KPT=KPT-1
    IF (KPT.LT.0) GO TO 110
    FLD(KPT,FP,CW(1,J))=FLD(K1,KPT,YY)
    KPT=KPT+K
110  FLD(34,35,SREG)=FLD(35,35,SREG)
    FLD(CU1,1,SREG)=FLD(0,1,SWI)
    CALL ENCODE(SREG,CL,K,HK1,HK2,YY)
    FLD(KPT,KPT,CW(1,J))=FLD(K1,KPT,YY)
    FLD(34,35,SWI)=FLD(35,35,SWI)
    KPT=KPT+K
    K1=(KPP-NBPSW)+1
    IF (K1) 150,150,100
150  KPT=K
    GO TO 110
160  KPK=K-KPT
    IF (KPK.EQ.0) GO TO 200
    KPK=KPK-1
    FLD(35,KPK,CW(1,J))=FLD(KPK1,KPK,YY)
200  CONTINUE

```

C (c) COMPLEX CHANNEL

```

MMIN=MINBK+1
M=3
BURST=0
DO 200 I=1,NW
  DO 200 J=1,K
    CV(I,J)=0
    IF (BURST.EQ.1) GO TO 201
    IF (I.GT.NW2) GO TO 200
    GO TO 203
201  FLD(LF1,LP,EV(I,J))=FLD(GBL1,LP,AA)
    GO 202 GG=1,BDP

```

Figure A.2 (cont.) Simulation computer program-hybrid system

ORIGINAL PAGE IS
OF POOR QUALITY

[illegible]

Figure A.2 (cont.) Simulation computer program-hybrid system


```

305 KNT=KNT-1
   CS=5+I
   KNT=50-KNT
   KNT1=KNT-1
   KNT2=KNT-1

   (1) VATERBI DECODER
   FLU(K1,KNT,VD)=FLU(KK1,KNT,Rn(15,JS))
   I1=1
   DO 450 J=1,N5
   I1=I1+1
   DO 400 I=1,IF,1H
   DFRMS=0
   DFRMS=NOT (TAGLE(I,J),VD)
   SKKNT=0
   DO 350 KNT=1,N
   I1 (FLU(I,J),DFRMS).NE.1) GO TO 320
   SKKNT1=SKKNT+1
   FLU(34,35,DFRMS)=FLU(35,35,DFRMS)
320 CONTINUE
350 SCORE(I,J)=SCORE(1)+SKKNT
   DO 110 I=1,N5
   I1=I1+1
   I1 (SCORE(I,J)-SKORE(IF,J)) 410,410,420
   SKORE(J)=SKORE(I1,J)
   CALL SS5(JLST,DCL,SS(1),SST)
   CALL SS5(JLST,DCL,SS(1),SST)
   DO 420 J=1,N5
   SKORE(J)=SKORE(IF,J)
   CALL SS5(JLST,DCL,SS(IF),SST)
   CALL SS5(JLST,DCL,SS(IF),SST)
   DO 450 J=1,N5
   I1=I1+1
   FLU(34,35,SS(J))=FLU(35,35,SS(J))
   DO 450 I=1,N5
   SKORE(I)=SS(J)
   SKORE(I)=SKORE(I)
   CONTINUE
450 CONTINUE

```

Figure A.2 (cont.) Simulation computer program-hybrid system


```

        GO 630 N=1,NBPS
        TI=TI+FLD(0,1,CER(1,J))
        BLI(1)=BLI(1)+FLD(0,1,CER(1,J))
        FLI(0,1,ITI)=FLD(0,1,CER(1,J))
        BLKI=BLKI+FLD(0,1,CER(1,J))
        IF (BLKI.NE.0) BLKI=BLKI+1
        IF (FLD(GLSK1,GDSK,ITI).NE.0) GO TO 624
015 GO 620 Z=1,12
        IF (BLKI-Z) 024,617,620
017 BLI(Z)=BLI(Z)+1
        FLI(Z)=FLI(Z)+BLKI
        GO TO 625
020 CONTINUE
        BLI(13)=BLI(13)+1
        FLI(13)=FLI(13)+BLKI
025 BLKI=-1*GDSK
        BLI=0
026 FLI(34,35,CER(1,J))=FLD(35,35,CER(1,J))
        FLI(35,35,ITI)=FLD(34,35,ITI)
        ITI=ITI+1
        IF (ITI-ITI0) 030,015,031
030 CONTINUE
031 BLI(1)=XOR(BLI(1),SI(1))
        GO 640 N=1,NBPS
        TI=TI+FLD(0,1,DER(1))
        BLI(1)=BLI(1)+FLD(0,1,DER(1))
        FLI(0,1,ITI)=FLD(0,1,DER(1))
        BLKI=BLKI+FLD(0,1,DER(1))
        IF (BLKI.NE.0) BLKI=BLKI+1
        IF (FLD(GLS1,GDS,ITI).NE.0) GO TO 644
035 GO 640 Z=1,12
        IF (BLKI-Z) 044,637,640
037 BLI(Z)=BLI(Z)+1
        FLI(Z)=FLI(Z)+BLKI
        GO TO 035
040 CONTINUE
        BLI(13)=BLI(13)+1
        FLI(13)=FLI(13)+BLKI
045 BLKI=-1*GDS
        BLI=0
044 FLI(34,35,DER(1))=FLD(35,35,DER(1))
        FLI(35,35,ITI)=FLD(34,35,ITI)
        ITI=ITI+1

```

Figure A.2 (cont.) Simulation computer program-hybrid system


```

NBLFI=NI
NBLFO=NO
DO 780 Z=1,12
NBLFI=NBLFI+NBLI(Z)*Z
NBLFO=NBLFO+NBL0(Z)*Z
780 CONTINUE
NRTI=NRTI
NRT0=NRT0
DO 790 Z=2,12
NRTI=NRTI-NBLI(Z)*(Z-1)
NRT0=NRT0-NBL0(Z)*(Z-1)
790 CONTINUE
IF (NBLI(13)*20.0) GO TO 792
NRTI=NRTI-NBLI(13)*((NBLFI/NBLI(13))-1)
792 IF (NBL0(13)*20.0) GO TO 794
NRT0=NRT0-NBL0(13)*((NBLFO/NBL0(13))-1)
794 DO 800 Z=1,13
NBLTI=NBLTI+NBLI(Z)
NBLTO=NBLTO+NBL0(Z)
BECTI=BECTI+BECI(Z)
BECTO=BECTO+BECO(Z)
PBBLI(Z)=(FLOAT(NBLI(Z))/FLOAT(NRTI))*100
PBBLI=PBBLI+PBBLI(Z)
PBBLI(Z)=(FLOAT(BECI(Z))/FLOAT(NBTI))*100
PBBLI=PBBLI+PBBLI(Z)
PBBL0(Z)=(FLOAT(NBL0(Z))/FLOAT(NRT0))*100
PBBL0=PBBL0+PBBL0(Z)
PBBL0(Z)=(FLOAT(BECO(Z))/FLOAT(NBTO))*100
PBBL0=PBBL0+PBBL0(Z)
800 CONTINUE
PNL01=(FLOAT(NEPW01)/FLOAT(NW2))*100
PNL00=(FLOAT(NEPW00)/FLOAT(NW2))*100
DO 801 Z=1,5
PNL1(Z)=(FLOAT(NEPW1(Z))/FLOAT(NW2))*100
PNL0(Z)=(FLOAT(NEPW0(Z))/FLOAT(NW2))*100
801 CONTINUE
PNLTI=(FLOAT(NEPW11)/FLOAT(NW2))*100
PNLTO=(FLOAT(NEPW10)/FLOAT(NW2))*100
POTPEO
DO 805 Z=1,12
PEL(Z)=PEL(Z)*100
805 POTP=POTP+PEL(Z)

```

Figure A.2 (cont.) Simulation computer program-hybrid system

1	0	.00000000
2	0	.00000000
3	0	.00000000
4	0	.00000000
5 OR MORE	0	.00000000
TOTAL	10	100.00000000

IMPROVEMENT OF THE CHANNEL DUE TO CODING -- IN TERMS OF ERROR PROB. PER BIT
 DECREASING OF ERROR PROB. (WITHOUT CODING - WITH CODING) IS 2.187500 %
 RATIO OF ERROR PROB. (WITH CODING / WITHOUT CODING) IS .000000

WF4H

Figure A.3 (cont.) Typical printout-hybrid system

ORIGINAL PAGE IS
 OF POOR QUALITY

REFERENCES

1. C. F. Shannon, "A Mathematical Theory of Communication," Bell System Technical Journal, Vol. 27, pp. 379-423, July 1948, pp. 623-656, October 1948.
2. M. J. E. Golay, "Notes on Digital Coding," Proc. IRE, Vol. 37, p. 657, June 1949.
3. P. Elias, "Coding for Noisy Channels," IRE Convention Record, part 4, pp. 37-47, 1955.
4. D. E. Muller, "Application of Boolean Algebra to Switching Circuit Design and to Error Detection," IRE Trans. Electron. Comput., Vol. EC-3, pp. 6-12, Sept. 1954; Math. Rev., Vol. 16, p. 99.
5. I. S. Reed, "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme," IRE Trans. Inform. Theory, Vol. IT-4, pp. 38-49, Sept. 1954; Math. Rev., Vol. 19, p. 721.
6. D. Slepian, "A Class of Binary Signaling Alphabets," Bell System Tech. J., Vol. 35, pp. 203-234, 1956. - "Some Further Theory of Group Codes," Bell System Technical Journal, pp. 1219-1252, 1960.
7. E. Prange, "Cyclic Error-Correcting Codes in Two Symbols," AFRCRTN - 57, 103, Air Force Cambridge Research Center, Cambridge, Massachusetts, Sept. 1957.
8. A. Hocquenghem, "Codes Correcteurs D'erreurs," Chiffres, 2, pp. 147-156, 1959.
9. R. C. Bose and D. K. Ray-Chandhuri, "On a Class of Error Correcting Binary Group Codes," Information and Control, 3, pp. 68-69, March 1960.
10. R. C. Bose and D. K. Ray-Chandhuri, "Further Results on Error Correcting Binary Group Codes," Information and Control, 3, pp. 279-290, Sept. 1960.
11. I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Field," J. Soc. Indust. Appl. Math., 8, pp. 300-304, June 1960.
12. D. Gorenstein and N. Zierler, "A Class of Error-Correcting Codes in p^m symbols," J. Soc. Ind. Appl. Math., 9, pp. 204-214, June 1961.
13. W. W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chandhuri Codes," IRE Trans. on Information Theory, IT-6, pp. 459-470, Sept. 1960.

REFERENCES (Continued)

14. L. D. Rudolph, "Geometric Configuration and Majority Logic Decodable Codes," MEE Thesis, University of Oklahoma, Norman, Oklahoma, 1964.
15. J. M. Wozencraft and B. Reiffen, Sequential Decoding, Technology Press and Wiley, New York, 1961.
16. R. M. Fano, "A Heuristic Discussion of Probabilistic Decoding," IEEE Trans. on Information Theory, IT-9, pp. 64-74, April 1963.
17. A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Trans. on Information Theory, IT-13, pp. 260-269, January 1967.
18. D. J. Costello, Jr., "Free Distance Bounds for Convolutional Codes," IEEE Trans. on Information Theory, Vol. IT-20, pp. 356-365, May 1974.
19. L. R. Bahl, C. D. Cullum, W. D. Frazer and F. Jelinek, "An Efficient Algorithm for Computing Free Distance," IEEE Trans. on Information Theory, Vol. IT-18, pp. 437-439, May 1972.
20. R. W. Boyd, "Hybrid Decoding of High Rate Convolutional Codes for Improved Burst Error Performance," Ph.D. Dissertation, Mississippi State University, 1976.
21. Shu Lin, An Introduction to Error-Correcting Codes, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1970.
22. G. D. Forney, Jr. "Convolutional Codes II: Maximum Likelihood Decoding," Information and Control, Vol. 25, pp. 222-266, July 1974.
23. F. Hemmati and D. J. Costello, Jr., "Truncation Error Probability in Viterbi Decoding," IEEE Trans. on Communications, Vol. Com-25, pp. 530-532, May 1977.
24. J. M. Wozencraft, Sequential Decoding for Reliable Communication, M.I.T., RLE Tech. Rept. TR 325, 1957.
25. A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems," Trans. on Communication Technology, Vol. Com-19, No.5, pp. 751-752, October 1971.
26. J. Odenwalder, "Optimal Decoding of Convolutional Codes," Ph.D. Dissertation, UCLA, 1970.

REFERENCES (Continued)

27. J. M. Wozencraft and I. M. Jacobs, Principles of Communication Engineering, John Wiley & Sons, Inc. New York, 1965.
28. K. S. Gilhousen, J. A. Heller, I. N. Jacobs and A. J. Viterbi, "Coding Systems Study for High Data Rate Telemetry Links," Linkabit Corporation, San Diego, California, January 1971.
29. W. W. Peterson and E. J. Weldon, Jr., Error-Correcting Codes, 2nd Ed., MIT Press, Cambridge, Massachusetts, 1972.
30. A. D. Wyner and R. B. Ash, "Analysis of Recurrent Codes," IEEE Trans. on Information Theory, IT-9, pp. 143-156, July 1963.
31. D. W. Hagelbarger, "Recurrent Codes: Easily Mechanized, Burst-Correcting, Binary Codes," Bell Systems Tech. J., 38, pp. 969-984, July 1959.
32. W. W. Peterson, Error-Correcting Codes, MIT Press, Cambridge, Massachusetts, 1961.
33. Y. Iwadare, "On Type B1 Burst-Error-Correcting Convolutional Codes," IEEE Trans. on Information Theory, IT-14, pp. 557-583, July 1968.
34. E. R. Berlekamp "Note on Recurrent Codes," IEEE Trans. on Information Theory, IT-10, pp. 257-258, July 1964.
35. J. A. Heller and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communication," IEEE Trans. on Comm. Tech., Vol. Com-19, No.5, pp. 835-848, October 1971.
36. J. W. Modestino, "Convolutional Code Performances in the Rician Fading Channel," IEEE Trans. on Comm., Vol. Com-24, No. 6, pp. 592-605, June 1976.
37. B. D. Trumpis and P. L. McAdams, "Performance of Convolutional Codes on Burst Noise Channels," National Telemetry Conference, December 1977.
38. K. L. Larsen, "Short Convolutional Codes with Maximal Free Distance for Rate $1/2$, $1/3$, and $1/4$," IEEE Trans. on Information Theory, pp. 371-372, May 1973.
39. F. M. Ingels, A Study of Digital Holographic Filter Generation, Dept. of Electrical Engineering, Mississippi State University.